I. Kießling / M. Lowes

Programmierung mit FORTRAN 77

```
PROGRAM  TEXT
CHARACTER T*1
READ *, T
IF (T.EQ.' ') THEN
   CALL SUBB(T)
ELSE IF (T.EQ.'A') THEN
   CALL SUBA(T)
ELSE
   CALL SUBS(T)
END IF
END
```

Teubner Studienskripten

Teubner Bücher DATENVERARBEITUNG / INFORMATIK

Bauknecht/Zehnder: Grundzüge der Datenverarbeitung
Methoden und Konzepte für die Anwendungen
286 Seiten. DM 24,80

Brauch: Programmierung mit BASIC
200 Seiten. DM 12,80

Brauch: Programmierung mit FORTRAN
5. Aufl. 224 Seiten. DM 14,80

Dal Cin: Fehlertolerante Systeme
Modelle der Zuverlässigkeit, Verfügbarkeit, Diagnose und Erneuerung
206 Seiten. DM 23,80

Görke: Fehlerdiagnose digitaler Schaltungen
230 Seiten. DM 15,80

Haacke u.a.: Datenverarbeitung für Ingenieure des Maschinenbaus und der Elektrotechnik
VIII, 315 Seiten. DM 38,--

Haase/Stucky/Wegner: Datenverarbeitung heute
284 Seiten. DM 21,80

Heinrich/Stucky: Programmierung mit ALGOL 60
2. Aufl. 157 Seiten. DM 12,80

Kaletsch: Programmierung mit PL/I
160 Seiten. DM 12,80

Kästner: Architektur und Organisation digitaler Rechenanlagen
224 Seiten. DM 22,80

Kießling/Lowes: Programmierung mit FORTRAN 77
184 Seiten. DM 12,80

Kupka/Wilsing: Dialogsprachen
168 Seiten. DM 21,80

Maurer: Datenstrukturen und Programmierverfahren
222 Seiten. DM 26,80

Mehlhorn: Effiziente Algorithmen
240 Seiten. DM 24,80

Menzel: BASIC in 100 Beispielen
2. Aufl. 216 Seiten. DM 21,80

Fortsetzung auf der 3. Umschlagseite

Zu diesem Buch

FORTRAN 77 ist die gebräuchliche Kurzbezeichnung der American National Standard Programming Language FORTRAN, definiert in dem Dokument ANSI X3.9-1978 und übernommen als neue deutsche Norm DIN 66 027. Diese Sprache ist als Nachfolger für die FORTRAN-Versionen gedacht, die auf dem ANSI-Standard X3.10-1966 basieren.

Durch die Aufnahme neuer Anweisungen zur strukturierten Programmierung, der Textverarbeitung und Dateiverwaltung wurden für FORTRAN 77 neue Anwendungsmöglichkeiten erschlossen oder der Gebrauch dieser Sprache wurde, verglichen mit älteren FORTRAN-Versionen, erleichtert.

In diesem Buch wurde eine vollständige Beschreibung des Sprachumfangs von FORTRAN 77 angestrebt. Die Erläuterung der Sprachelemente an vielen Beispielen sollen auch Lesern mit geringen Vorkenntnissen die Erarbeitung des "neuen" FORTRAN ermöglichen.

Programmierung mit FORTRAN 77

Von Dr. rer. nat. Immo Kießling
Akad. Oberrat an der Universität Göttingen

und Dipl.-Math. Martin Lowes
Wiss. Mitarbeiter an der Universität Göttingen

B. G. Teubner Stuttgart 1982

Dr. rer. nat. Immo Kießling, akad. Oberrat
Studium der Mathematik und Physik an den Universitäten München und Freiburg i.Br.. 1965 wissenschaftlicher Mitarbeiter der Philips Zentrallaboratorium GmbH Hamburg. 1968 Dozent an der Staatl. Ingenieurschule/FH Offenburg. Seit 1972 am Institut für Numerische und Angewandte Mathematik der Universität Göttingen.

Dipl.-Math. Martin Lowes
Studium der Mathematik und Physik an der Universität Göttingen, dort seit 1975 als wissenschaftlicher Angestellter am Institut für Numerische und Angewandte Mathematik mit der Durchführung von Programmierkursen beauftragt.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Kiessling, Immo:
Programmierung mit FORTRAN 77 / von Immo Kiessling u. Martin Lowes. - Stuttgart : Teubner, 1982.
(Teubner Studienskripten ; 89 : Datenverarbeitung)
ISBN 3-519-00089-X

NE: Lowes, Martin:; GT

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, besonders die der Übersetzung, des Nachdrucks, der Bildentnahme, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege, der Speicherung und Auswertung in Datenverarbeitungsanlagen, bleiben, auch bei Verwertung von Teilen des Werkes, dem Verlag vorbehalten.

Bei gewerblichen Zwecken dienender Vervielfältigung ist an den Verlag gemäß § 54 UrhG eine Vergütung zu zahlen, deren Höhe mit dem Verlag zu vereinbaren ist.

© B. G. Teubner, Stuttgart 1982

Printed in Germany
Gesamtherstellung: Beltz Offsetdruck, Hemsbach/Bergstr.
Umschlaggestaltung: W. Koch, Sindelfingen

Vorwort

FORTRAN ist, verglichen mit anderen Programmiersprachen, eine sehr "alte" und doch gleichzeitig sehr moderne Sprache: Das große Interesse vieler Anwender auf der einen und die Entwicklung der technischen Möglichkeiten auf der anderen Seite haben immer wieder dazu geführt, daß neue Sprachelemente aufgenommen und vorhandene erweitert wurden. Bei einer "lebenden" Sprache wie FORTRAN ist es nur natürlich, daß sich schnell Dialekte bilden, daß deshalb immer wieder neu allgemein gültige Regeln festgelegt werden müssen.

Die Bestrebungen nach Normierung führten 1978 nach etwa zehnjähriger Arbeit zur Veröffentlichung der Norm X3.9-1978 durch das American National Standards Institut (ANSI), durch welche die alte Norm X3.9-1966 ersetzt wurde. Die Sprache, die die Norm beschreibt, wird allgemein als "FORTRAN 77" bezeichnet. Sie ist in zwei Stufen definiert: Neben dem vollen Sprachumfang gibt es einen "Subset", in dem einige Sprachelemente ganz fehlen und andere nur eingeschränkt vorhanden sind. Dieses Buch behandelt den vollen Sprachumfang; die Einschränkungen sind in einem Anhang aufgeführt.

Vor allem in drei Punkten unterscheidet sich die FORTRAN 77-Norm wesentlich von ihrer Vorgängerin:

(1) Das Konzept der Block IF-Anweisungen erlaubt die Erstellung strukturierter Programme.
(2) Dem zunehmenden Bedürfnis, im Rahmen numerischer Problemstellungen auch Texte zu verarbeiten, trägt die Einführung des neuen Datentyps CHARACTER Rechnung.
(3) Für die Dateiverwaltung, die bislang bewußt vollständig ausgespart worden war, werden besondere Anweisungen zur Verfügung gestellt.

Die Norm legt grundsätzlich nur einen Rahmen dessen fest, was ein "processor" akzeptieren muß, während die Einzelheiten vielfach offen gelassen werden. So müssen "Textkonstanten" akzeptiert werden - aus wie vielen Zeichen sie bestehen dürfen, wird nicht gesagt. In diesem Buch sind in solchen Fällen Schranken, soweit bekannt, für verschiedene Rechner angegeben. Verzichtet wird dagegen weitgehend

auf die Beschreibung von Erweiterungen einzelner Hersteller; diese können dem Programmierer die Arbeit zwar gelegentlich erleichtern, schränken jedoch gleichzeitig die Portabilität von Programmen stark ein.

Unser Dank gilt den Firmen CDC und IBM, die freundlicherweise Beschreibungen ihrer FORTRAN 77-Versionen zur Verfügung stellten. Unser Dank gilt weiter der Gesellschaft für Wissenschaftliche Datenverarbeitung Göttingen (GWDG) und dem Institut für Numerische und Angewandte Mathematik der Universität Göttingen, auf deren Anlagen UNIVAC 1100/83 bzw. VAX-11 wir die Möglichkeiten zum Testen der Beispiele erhielten. Dem Verlag B.G.Teubner gilt unser Dank für die verständnisvolle Zusammenarbeit.

Göttingen, Januar 1982

Immo Kießling
Martin Lowes

Inhaltsverzeichnis

1	Einleitung	11
1.1	Einige Grundbegriffe	11
1.2	Darstellung nichtnumerischer und numerischer Daten	13
1.3	Formalismus zur Beschreibung von Anweisungen	16
1.4	FORTRAN-Anweisungen	17
1.5	Reihenfolge der Anweisungen	18
2	Grundelemente von FORTRAN	19
2.1	Arithmetische Daten und ihre Verarbeitung	19
2.1.1	Arithmetische Konstante	19
2.1.2	Symbolischer Name	20
2.1.3	Variable	21
2.1.4	Symbolische Namen von Konstanten	21
2.1.5	IMPLICIT und Typanweisung	22
2.1.6	Arithmetischer Ausdruck	24
2.1.7	Arithmetische Wertzuweisung	27
2.1.8	Arithmetisches IF	27
2.2	Das vollständige Programm	28
2.2.1	PROGRAM	28
2.2.2	Listengesteuerte Ein-/Ausgabe	29
2.2.3	END, STOP, PAUSE	31
2.2.4	Ein FORTRAN-Programm als Anwendungsbeispiel	32
2.3	GOTO	35
2.4	Felder	37
2.5	DO	41
2.6	CONTINUE	46
3	Arbeiten mit logischen Daten	47
3.1	Vergleichsausdrücke	47
3.2	Logischer Ausdruck	48
3.3	Logische Wertzuweisung	49
3.4	Logische IF-Anweisung	49
3.5	Block IF, END IF, ELSE IF, ELSE	50
4	Unterprogramme	59
4.1	Funktionen	60

4.1.1	Innere Standardfunktionen	61
4.1.2	Anweisungsfunktionen	63
4.1.3	Externe Funktionen	66
4.2	Subroutine-Unterprogramme	70
4.3	Formale und aktuelle Parameter	71
4.3.1	Ein- und Ausgabeparameter, Nebeneffekte	72
4.3.2	Felder als Parameter	74
4.3.3	Prozeduren als Parameter	78
4.4	Außergewöhnlicher Rücksprung	82
4.5	ENTRY	83
5	Textverarbeitung	86
5.1	Textvariable und Textkonstanten	86
5.1.1	Das Schlüsselwort CHARACTER	86
5.1.2	Textkonstanten	88
5.1.3	Teilketten	89
5.2	Arbeiten mit Texten	91
5.2.1	Textausdrücke	91
5.2.2	Wertzuweisung	91
5.2.3	Vergleiche von Texten	92
5.2.4	Die Länge (*)	94
5.2.5	Beispiele	95
5.2.6	Texte als Parameter von Unterprogrammen	98
6	Arbeiten mit Dateien	101
6.1	Grundbegriffe	101
6.1.1	Sätze	101
6.1.2	Dateien	102
6.1.3	Dateinummern	103
6.1.4	Zugriff und Positionierung	104
6.1.5	Dateiende	106
6.2	Die Anweisungen	106
6.2.1	Aufbau der Anweisungen	107
6.2.2	Der Steuerparameter UNIT	108
6.2.3	Behandlung von Fehlern	109
6.2.4	Erkennen des Dateiendes	110
6.2.5	Ein-/Ausgabelisten	110

6.3	OPEN	112
6.3.1	Der Steuerparameter FILE	112
6.3.2	Der Steuerparameter STATUS	114
6.3.3	Die Parameter ACCESS, FORM, RECL und BLANK	115
6.4	Datenübertragung	116
6.4.1	READ und WRITE	116
6.4.2	Kurzformen	117
6.5	Positionierung sequentieller Dateien	118
6.6	CLOSE	121
6.7	INQUIRE	122
6.7.1	Die Parameter EXIST, OPENED, NUMBER, NAMED und NAME	124
6.7.2	Die Parameter ACCESS, SEQUENTIAL und DIRECT	124
6.7.3	Die Parameter FORM, FORMATTED und UNFORMATTED	125
6.7.4	Die Parameter RECL und NEXTREC	125
6.7.5	Der Parameter BLANK	126
7	Formatierung	127
7.1	Formatgesteuerte Umwandlung	127
7.1.1	Formatbeschreiber	128
7.1.2	Positionierung	129
7.1.3	Eingabe numerischer Werte	131
7.1.4	Ausgabe numerischer Werte	133
7.1.5	Ein-/Ausgabe logischer Werte	136
7.1.6	Ein-/Ausgabe von Texten	137
7.1.7	Drucken	140
7.1.8	Wiederholungsfaktoren	142
7.1.9	Zusammenhang zwischen Ein-/Ausgabeliste und Format	142
7.1.10	Skalenfaktoren	146
7.2	Listengesteuerte Umwandlung	147
7.2.1	Listengesteuerte Eingabe	147
7.2.2	Leere Eingabefelder	149
7.2.3	Wiederholungsfaktoren	150
7.2.4	Listengesteuerte Ausgabe	150
7.3	Interne Dateien	151
8	Speicherplatzorganisation und Anfangswerte	154
8.1	COMMON	154

8.2 EQUIVALENCE 159
8.3 DATA 160
8.4 BLOCK DATA 162
8.5 SAVE 163

Anhang A: ASCII- und EBCDIC-Code 166

Anhang B: Innere Standardfunktionen 168

Anhang C: Restriktionen des Subset 176

Literaturverzeichnis 180

Stichwortverzeichnis 181

1 Einleitung

1.1 Einige Grundbegriffe

Ein ausführbares Programm besteht aus einem Hauptprogramm und aus beliebig vielen Unterprogrammen. Jedes Hauptprogramm und jedes Unterprogramm bildet für sich je eine Programmeinheit. Jede Programmeinheit besteht aus einer Folge von Anweisungen, die durch Anweisungsnummern gekennzeichnet sein können, und aus Kommentarzeilen.

Äußerlich betrachtet besteht ein ausführbares Programm aus Zeilen, die Ketten von Zeichen (characters) enthalten. Die Norm schreibt vor, daß der FORTRAN-Zeichensatz zur Verfügung stehen muß. Dieser enthält die 26 Buchstaben

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

die 10 Ziffern

0 1 2 3 4 5 6 7 8 9

und die 13 Sonderzeichen

␣ = + - * / ( ) , . $ ' :

wo ␣ das Leerzeichen bedeutet. Wenn keine Mißverständnisse zu befürchten sind, wird es in diesem Text auch durch eine Lücke gekennzeichnet. Das Währungszeichen $ darf durch ein anderes Währungszeichen ersetzt sein. Im übrigen ist der FORTRAN-Zeichensatz als kleinster Durchschnitt der Zeichenvorräte der existierenden Rechner zu betrachten. Diese operieren im allgemeinen auf den wesentlich umfangreicheren Zeichenvorräten bekannter Codes wie dem ASCII-Code, dem EBCDIC-Code usw..

Eine Zeile umfaßt 72 Zeichenpositionen, numeriert durch 1,2,...,72, die als Spalten bezeichnet werden. Andere Zeichenpositionen sind bedeutungslos, so beispielsweise die Spalten 73 bis 80 in 80-spaltigen Lochkarten. Sie werden vom Compiler ignoriert und können daher Beliebiges enthalten.

Eine Anweisung steht in einer oder in mehreren Zeilen. Die erste Zeile enthält in Spalte 6 eines der Zeichen ␣ oder 0. Die zweite, dritte,... Zeile heißt Fortsetzungszeile. Eine solche ist gekenn-

zeichnet durch ein beliebiges von 0 und ß verschiedenes Zeichen in Spalte 6. Dabei müssen die Spalten 1 bis 5 leer sein. Es sind bis zu 19 Fortsetzungszeilen zulässig.

Es werden ausführbare (executable) und nicht ausführbare Anweisungen (nonexecutable statements) unterschieden. Ausführbare Anweisungen beschreiben Handlungen, die bei der Abarbeitung eines Programms vollzogen werden. Nicht ausführbare Anweisungen enthalten Information für den Compiler, z.B. zur Organisation von Speicherplatz.

Jede Anweisung kann in den Spalten 1 bis 5 ihrer ersten Zeile durch eine Anweisungsnummer gekennzeichnet sein. Sie besteht aus höchstens 5 Ziffern, von denen mindestens eine von 0 verschieden ist. Leerzeichen und führende Nullen sind bedeutungslos. Eine Anweisungsnummer kann insoweit beliebig in den Spalten 1 bis 5 der ersten Zeile jeder Anweisung stehen. Anweisungsnummern ausführbarer Anweisungen werden in diesem Buch Marken genannt. Sie können in einigen Steueranweisungen (siehe 2.1.8, 2.3) als Sprungziele verwendet werden. Nur der Vollständigkeit halber ist hier als Ausnahme zu vermerken, daß Marken der ausführbaren Anweisungen ELSE IF und ELSE (siehe 3.5) nicht als Sprungziele verwendet werden dürfen.

Eine Kommentarzeile enthält eines der Zeichen C oder *(Stern) in Spalte 1 oder nur Leerzeichen in den Spalten 1 bis 72. Steht C oder * in Spalte 1, so können die Spalten 2 bis 72 beliebige Zeichen enthalten.

Kommentarzeilen dienen nur der Dokumentation und beeinflussen ein ausführbares Programm überhaupt nicht. Sie können daher an beliebiger Stelle in jeder Programmeinheit erscheinen, nur nicht in der letzten: In dieser steht immer die Anweisung

```
END
```

Sie allein ist bereits das kürzeste aller ausführbaren Programme, allerdings ein nicht sehr sinnvolles.

Die Anweisung END steht nicht nur immer, sondern auch ausschließlich in der letzten Zeile jeder Programmeinheit. Sie kennzeichnet dadurch einmal für den Compiler das Ende einer Programmeinheit. Andererseits gehört sie zu den ausführbaren Anweisungen. Ihre Aus-

führung in einem Hauptprogramm bewirkt dessen Beendigung, in einem Unterprogramm eine Rückverzweigung in die übergeordnete Programmeinheit, die das Unterprogramm aufrief.

Das Wort END gibt somit anschaulich die Bedeutung einer Anweisung wieder. Es gehört zu den Schlüsselwörtern, die allgemein die Wirkung von Anweisungen mnemotechnisch umschreiben, wie z.B. READ, PRINT, CONTINUE, STOP, END.

Ohne auf Einzelheiten der entsprechenden Anweisungen einzugehen, kann die Form eines FORTRAN-Programms an einem kurzen selbsterklärenden Beispiel erläutert werden.

```
C DIE READ-ANWEISUNG LIEST 'KONSTANTEN' ZU DEN
C 'VARIABLEN' MIT 'NAMEN' A, B
      READ*,A,B
C DIE PRINT-ANWEISUNGEN DRUCKEN DEN WERT DES
C PRODUKTS A*B BZW. 'TEXTKONSTANTEN'
      PRINT*,'A*B=',A*B
      PRINT*,'DAS IST NUR EIN BEISPIEL'
            E  N  D
```

Eine Anweisung steht in ihrer ersten Zeile und *ggf.* in ihren Fortsetzungszeilen in den Spalten 7 bis 72. Leerzeichen sind bedeutungslos, wenn sie nicht zu einer sogenannten Textkonstanten gehören, d.h., sie können eingefügt oder weggelassen werden. Sie können verwendet werden, um die Lesbarkeit eines Programms zu verbessern. Die Anweisung END muß in einer Zeile stehen. Für sie ist also keine Fortsetzungszeile zulässig.

1.2 Darstellung nichtnumerischer und numerischer Daten

Die Norm nennt hinsichtlich der internen Darstellung von Daten nur Rahmenbedingungen.

Meistens werden nichtnumerische Daten aus den Zeichenvorräten des ASCII-Code oder des EBCDIC-Code gebildet. Die Zeichen dieser Codes sind im Anhang A wiedergegeben.

Die Norm fordert, daß ein Zeichen in einer Speichereinheit für ein

Zeichen dargestellt wird. Da der sprachliche Gebrauch dieses Begriffs umständlich ist, wird er in diesem Buch durch die Kurzform Zeichenposition oder auch nur Zeichen ersetzt. Der Kontext verhindert Mißverständnisse. Der ASCII-Code ist ein 7-Bit-, der EBCDIC-Code ein 8-Bit-Code. Ein Zeichen umfaßt bei Verwendung dieser Codes mindestens 7 bzw. 8 Bits. Es können redundante Bits zur Fehler-Erkennung oder -Korrektur vorgesehen sein.

Im Zusammenhang mit der Darstellung numerischer Daten führt FORTRAN den Begriff der numerischen Speichereinheit ein. In diesem Buch wird dafür die sonst übliche griffige Bezeichnung Wort gebraucht. Es müssen unbestimmte Teilmengen der ganzen, reellen und komplexen Zahlen zur Verfügung stehen, die in je einem oder in je zwei Worten dargestellt werden. Weitere Vorschriften gibt es nicht. Etwas anderes ist auch gar nicht möglich, weil die Darstellung numerischer Daten stark anlagenabhängig ist. Im Rahmen dieses Buches ist es deshalb nicht sinnvoll, Beispiele für die Darstellung von Daten anzugeben.

Gebräuchliche Wortlängen sind 32, 36 oder auch 60 Bit. Damit hängt es auch von der Anlage ab, wieviele Zeichen pro Wort maximal gespeichert werden können.

Logische Daten kommen z.B. ins Spiel, wenn Vergleichsausdrücke wie "5 < 1" oder "7 > 1" erscheinen. Wenn man einen solchen Ausdruck als Aussage auffaßt, ist es naheliegend, ihm den Wahrheitswert "wahr" oder "falsch" zu erteilen. Zur internen Darstellung dieser Werte genügt es, ein Bit vorzusehen.

FORTRAN kennt sechs Datentypen (kurz: Typen)

```
CHARACTER
INTEGER
REAL
DOUBLE PRECISION
COMPLEX
LOGICAL
```

Die Zuordnung der Schlüsselwörter zu den verschiedenen Datenarten ist selbsterklärend.

Die Datentypen ermöglichen es, zu jedem Datum die entsprechende Vorschrift auszuwählen, welche die Abbildung zwischen dem Datum und seiner internen Darstellung vermittelt.

Jedes INTEGER- und jedes REAL-Datum belegt je ein Wort ebenso wie jedes LOGICAL-Datum, obwohl dafür ein Bit ausreichen würde. DOUBLE PRECISION- und COMPLEX-Daten belegen jeweils zwei konsekutive Worte im Hauptspeicher.

Für Größen der Typen REAL und DOUBLE PRECISION wird gelegentlich der Sammelbegriff Gleitkomma-Größen gebraucht. Daten der Typen REAL bzw. DOUBLE PRECISION werden manchmal als einfach- bzw. doppeltgenaue Gleitkomma-Daten bezeichnet. Daten der Typen INTEGER, REAL, DOUBLE PRECISION sind reellwertig.

In Erweiterung der Norm kennen manche FORTRAN-Versionen für Zahlen weitere Datentypen, beispielsweise COMPLEX-Zahlen doppelter Länge in FORTRAN(ASCII) von UNIVAC und VAX-11 FORTRAN von DEC. Alternativ zu den Schlüsselwörtern der Norm werden auch Schlüsselwörter gebraucht, die mit einer Längenangabe versehen sind. Diese mißt den Speicherplatzbedarf eines Datums des jeweiligen Typs durch die entsprechende Anzahl Speichereinheiten für ein Zeichen. Man hat dann beispielsweise folgendes:

Norm	UNIVAC, DEC
REAL	REAL*4
DOUBLE PRECISION	REAL*8
COMPLEX	COMPLEX*8
-	COMPLEX*16

Bei DEC können weiter INTEGER und INTEGER*4 alternativ mit gleicher Bedeutung gebraucht werden und zusätzlich gibt es INTEGER*2, d.h. also abweichend von der Norm ganze Zahlen in einer halben numerischen Speichereinheit, oder besser gesagt, in zwei Bytes, weil die VAX-11 eine byteorientierte Maschine ist.

1.3 Formalismus zur Beschreibung von Anweisungen

Sonderzeichen des FORTRAN-Zeichensatzes, Großbuchstaben und daraus gebildete Ketten sind wie angegeben zu verwenden.

Kleinbuchstaben und daraus gebildete Ketten sind durch Größen des jeweiligen Programms zu ersetzen.

Eckige Klammern [] schließen Angaben ein, die wahlweise geschrieben oder weggelassen werden können.

Punkte ... bedeuten, daß die vorangehenden in eckige Klammern eingeschlossenen Angaben beliebig oft wiederholt werden können.

Leerzeichen dienen nur zur Verbesserung der Lesbarkeit und haben, ausgenommen in Textkonstanten, keine Bedeutung.

Beispielsweise wird die "Typanweisung" wie folgt angegeben:

```
t  v[,v]...
```

Hierin bedeuten:

t — Datentyp INTEGER, REAL, DOUBLE PRECISION, COMPLEX oder LOGICAL

v — eine vom Benutzer nach bestimmten Vorschriften zu bildende Zeichenkette mit bestimmter Bedeutung, z.B. "Name einer Variablen" wie A oder B in dem Programm-Beispiel von Abschnitt 1.1

Wenn A,B "Variable" vom Typ INTEGER benennen sollen, könnte das Programm die Typanweisung

```
INTEGER A,B
```

enthalten.

1.4 FORTRAN-Anweisungen

In der folgenden Tabelle werden alle FORTRAN-Anweisungen genannt.

Nicht ausführbare Anweisungen	Ausführbare Anweisungen
Spezifikationsanweisungen	Wertzuweisungen
DIMENSION	arithmetische
EQUIVALENCE	logische
COMMON	für Texte
Typanweisungen	ASSIGN
IMPLICIT	
PARAMETER	Steueranweisungen
EXTERNAL	
INTRINSIC	GOTO
SAVE	IF, arithmetisches
	IF, logisches
Erteilung von Anfangswerten	Block IF
	ELSE IF
DATA	ELSE
	END IF
Format-Anweisung	DO
	CONTINUE
FORMAT	STOP
	PAUSE
Anweisungsfunktion	END
	CALL
Definition einer Anweisungsfunktion	RETURN
Benennung von Haupt- und Unterprogrammen	Ein-/Ausgabe
	READ
PROGRAM	WRITE

FUNCTION
SUBROUTINE
ENTRY
BLOCK DATA

PRINT
OPEN
CLOSE
INQUIRE
BACKSPACE
ENDFILE
REWIND

1.5 Reihenfolge der Anweisungen

Die Anweisungen dürfen nicht in willkürlicher Reihenfolge in einer Programmeinheit erscheinen. Die meisten Regeln über ihre Ordnung können aus dem nachstehenden Schema abgelesen werden. Die Reihenfolge von Anweisungen, die durch horizontale Linien getrennt sind, muß in jeder Programmeinheit mit der des Schemas übereinstimmen. Zusätzliche Vorschriften gibt es für die Anweisungen PARAMETER und ENTRY. Sie ergeben sich aus den Bedeutungen der Anweisungen und werden daher in den Abschnitten 2.1.4 und 4.5 angegeben.

<table>
<tr><td rowspan="6">Kommentar-
zeilen</td><td colspan="3">PROGRAM, FUNCTION, SUBROUTINE
BLOCK DATA</td></tr>
<tr><td rowspan="2">FORMAT

ENTRY</td><td rowspan="2">PARAMETER</td><td>IMPLICIT</td></tr>
<tr><td>Andere Spezifikationsanweisungen</td></tr>
<tr><td rowspan="2">DATA</td><td>Anweisungsfunktionen</td></tr>
<tr><td>ausführbare Anweisungen</td></tr>
<tr><td colspan="3">END</td></tr>
</table>

2 Grundelemente von FORTRAN

2.1 Arithmetische Daten und ihre Verarbeitung

2.1.1 Arithmetische Konstante

Die externen Darstellungen von Zahlen heißen arithmetische Konstanten; die Zahlen selbst, aufgefaßt als mathematische Objekte, sind Werte von arithmetischen Konstanten.

Eine INTEGER-Konstante ist eine Kette von Ziffern, der ein positives oder negatives Vorzeichen vorangehen kann. Sie wird als ganze Dezimalzahl aufgefaßt. Die Konstante einer negativen Zahl muß natürlich das negative Vorzeichen besitzen. Konstanten positiver ganzer Zahlen dürfen das positive Vorzeichen enthalten.

Von der jeweiligen Anlage hängt es ab, welcher Abschnitt der ganzen Zahlen intern darstellbar ist und welche INTEGER-Konstanten somit zur Verfügung stehen.

Die REAL-Konstante kann in drei Formen auftreten, nämlich als

(1) Kette von Ziffern mit einem Punkt an einer beliebigen Position, d.h. der Punkt kann auch das erste oder das letzte Zeichen sein, ein positives oder negatives Vorzeichen kann vorangehen. Der Punkt hat die Bedeutung eines Dezimalpunkts. Die Länge der Kette ist anlagenabhängig.

(2) REAL-Konstante wie in (1), gefolgt von einem Exponententeil, bestehend aus dem Buchstaben E, gefolgt von einer INTEGER-Konstanten, die ein positives oder negatives Vorzeichen haben kann. Der Exponententeil bedeutet eine Zehnerpotenz. Welcher Abschnitt der ganzen Zahlen Exponent sein kann, ist anlagenabhängig.

(3) INTEGER-Konstante, gefolgt von einem Exponententeil wie in (2).

Der Wert der REAL-Konstanten der Formen (2) und (3) ist das Produkt der vorangehenden Konstanten und der Zehnerpotenz, die durch den Exponententeil repräsentiert wird. Intern werden in einem Wort eine Mantisse m und ein Exponent z anlagenabhängig in verschiedener Weise dargestellt, so daß $m*b^z$ bis auf Rundungsfehler die Werte von

REAL-Konstanten ergibt. Die Basis b ist meistens 2 oder 16. Die Länge der Mantisse, gemessen in der Anzahl ihrer Bits, beschränkt die Anzahl der Dezimalstellen der Formen (1) und (2). Die Norm läßt es zu, in der Form (3) die INTEGER-Konstante auch mit mehr Dezimalstellen zu schreiben, als der Mantissenlänge entspricht. Diese Möglichkeit ist nicht durch alle Compiler realisiert.

Beipiele:

Zu (1):	2.	.501	-0.732	82.6537
Zu (2) bzw. (3):	3.2E2	-.4731E-03	2E1	.8265317E2

Die DOUBLE PRECISION-Konstante hat eine größere Mantissenlänge als die REAL-Konstante. Damit ist diese Zahlendarstellung von größerer, nicht notwendig von doppelter, Genauigkeit. Der zulässige Zahlenbereich ist anlagenabhängig. DOUBLE PRECISION-Konstanten gibt es in zwei Formen. Sie werden entsprechend den Formen (2) und (3) der REAL-Konstanten gebildet und von dieser dadurch unterschieden, daß im Exponententeil der Buchstabe D anstelle des Buchstabens E erscheint.

Beispiele: 1.D0 .35675D-3 100D0

Die COMPLEX-Konstante wird in der Form (r,i) dargestellt, wo r und i REAL- oder INTEGER-Konstanten sind, die den Real- bzw. Imaginärteil der komplexen Zahl ergeben.

Beipiele: (0.,1.) (-38.6E-3,11.1E2) (1,1)

Intern werden der Realteil und der Imaginärteil einer COMPLEX-Konstanten wie REAL-Konstanten dargestellt, auch wenn sie als INTEGER-Konstanten angegeben werden.

In alle arithmetischen Konstanten dürfen Leerzeichen eingestreut werden, ohne daß dies eine Bedeutung für ihren Wert hat.

Die Vorschriften für Konstanten zu nicht normgemäßen Datentypen müssen den Benutzerhandbüchern entnommen werden.

2.1.2 Symbolischer Name

Ein symbolischer Name (kurz: Name) ist eine Kette von höchstens 6

Buchstaben und/oder Ziffern, beginnend mit einem Buchstaben, und kann, außer anderen Größen, die später erklärt werden, eine Variable benennen.

2.1.3 Variable

Eine Variable ist ein Paar, bestehend aus einem symbolischen Namen und einem Datentyp. Durch Aufruf des symbolischen Namens einer Variablen wird auf ein Wort oder zwei Worte zugegriffen, welche die interne Darstellung der zugehörigen Konstanten enthalten. Um diese richtig deuten zu können, muß dem Compiler der entsprechende Datentyp bekannt sein. Dies begründet die obige Definition der Variablen. Wenn nichts anderes bestimmt ist, gilt: Der Typ einer Variablen ist INTEGER, wenn ihr Name mit einem der Buchstaben I, J, K, L, M oder N beginnt, sonst REAL. Diese Vorschrift wird häufig als Namensregel bezeichnet.

Einer Variablen kann ein Wert erteilt werden, der während der Ausführung eines Programms beliebig verändert werden kann.

Da der Begriff "symbolischer Name einer Variablen" sprachlich umständlich ist, wird im folgenden kurz Variable gesagt. Der Kontext verhindert dabei Mißverständnisse.

2.1.4 Symbolische Namen von Konstanten

Symbolische Namen von Konstanten werden definiert in der Anweisung

```
PARAMETER(p=e[,p=e]...)
```

Hierin sind

p symbolischer Name einer Konstanten
e konstanter Ausdruck

Ein Spezialfall eines konstanten arithmetischen Ausdrucks ist eine arithmetische Konstante. Die allgemeine Definition wird in Abschnitt 2.1.6 angegeben.

Durch die PARAMETER-Anweisung wird innerhalb einer Programmeinheit

als Stellvertreter für e ein symbolischer Name eingeführt. Derselbe Name darf nur einmal innerhalb einer Programmeinheit definiert werden.

Die PARAMETER-Anweisung ist von Vorteil, wenn der Wert des Ausdrucks e an mehreren Stellen einer Programmeinheit benötigt wird. Bei einer eventuellen Änderung von e muß diese dann nur in der PARAMETER-Anweisung vorgenommen werden.

Bei der Abarbeitung eines Programms steht p immer für denselben Wert, im Unterschied zum Namen einer Variablen.

Der "symbolische Name einer Konstanten" wird oft als benannte Konstante bezeichnet.

Wenn für eine benannte Konstante nicht die Namensregel gelten soll, so muß ihr Typ in einer IMPLICIT- oder Typanweisung (siehe 2.1.5) bestimmt werden, die vor der definierenden PARAMETER-Anweisung steht.

2.1.5 IMPLICIT und Typanweisung

Da durch die Namensregel nur die Typen INTEGER und REAL erfaßt werden, gibt es weitere Möglichketen der Typbestimmung.

Durch die IMPLICIT-Anweisung erfolgt eine Zuordnung von Buchstaben zu Typen. Alle Namen, die mit einem der dort genannten Buchstaben beginnen, sind von entsprechendem Typ. Die allgemeine Form dieser Anweisung ist

```
IMPLICIT t (a[,a]...)[,t(a[,a]...)]...
```

Hierin sind

t — Typ. Außer den arithmetischen Typen können auch LOGICAL (siehe 3) sowie CHARACTER[*1] (siehe 5.1.1) erscheinen

a — ein einzelner Buchstabe oder ein Abschnitt von Buchstaben des Alphabets, der durch seinen ersten und letzten Buchstaben gekennzeichnet wird, verbunden durch einen Bindestrich. Beispielsweise ist A-H gleichbedeutend mit A,B,C,D,E,F,G,H

Die Typbestimmung durch IMPLICIT-Anweisung geht der Namensregel vor.

Beipiel: IMPLICIT DOUBLE PRECISION(A-H), INTEGER(P,Q)

Hier werden Namen, die mit einem der Buchstaben A,B,C,...,H beginnen, mit dem Typ DOUBLE PRECISION verbunden, Namen, die mit P oder Q beginnen, mit dem Typ INTEGER.

In einer Programmeinheit darf mehr als eine IMPLICIT-Anweisung vorhanden sein, aber derselbe Buchstabe darf darin nicht mehrmals angesprochen werden. Übrigens sind nicht alle Namen mit einem Typ verbunden, da es außer Daten noch andere Größen gibt. Nicht alle Ketten von Buchstaben und Ziffern, welche die Bildungsregel für Namen erfüllen, sind Namen. Beispielsweise werden READ, PRINT, END als Schlüsselwörter verwendet. Allerdings ist es möglich, wenn auch nicht empfehlenswert, mit Schlüsselwörtern identische Namen zu definieren. Der Kontext ermöglicht dann dem Compiler die Unterscheidung zwischen Namen und Schlüsselwörtern.

Die Zuweisung eines Typs zu einem Namen auf Grund seines Anfangsbuchstabens kann übersteuert werden durch die Typanweisung

```
t v[,v]...
```

Hierin sind

t Typbezeichnung. Außer den arithmetischen Typen kann auch der Typ LOGICAL (siehe 3) erscheinen. Die Typanweisung für den Typ CHARACTER kann eine allgemeinere Form haben (siehe 5.1.1)

v Variablen, benannte Konstanten sowie Felder, Felddefinitionen (siehe 2.4) oder Funktionen (siehe 4.1.2, 4.1.3, 4.5)

Beispiele:

```
INTEGER A,B,SUMME
DOUBLE PRECISION X,Y,Z
COMPLEX V,W
```

Die PARAMETER-, IMPLICIT- und die Typanweisung gehören zu den Spezifikationsanweisungen. Diese dienen u.a. der Typbestimmung und der Speicherplatzreservierung. Sie sind nicht ausführbar und gehen al-

len ausführbaren Anweisungen voran. In einer Programmeinheit müssen IMPLICIT-Anweisungen allen anderen Spezifikationsanweisungen vorangehen mit Ausnahme von PARAMETER-Anweisungen, die vor oder hinter IMPLICIT-Anweisungen stehen können. Wenn jedoch der Typ einer benannten Konstanten in einer IMPLICIT-Anweisung bestimmt wird, so muß diese vor der PARAMETER-Anweisung stehen, welche die Konstante definiert.

Die Bestimmung des Typs einer Größe wird als Vereinbarung bezeichnet. Später wird dieser Begriff auch allgemein im Sinne einer Definition gebraucht.

2.1.6 Arithmetischer Ausdruck

Spezialfälle arithmetischer Ausdrücke sind arithmetische Konstanten, arithmetische Variablen, benannte arithmetische Konstanten, arithmetische Feldelemente (siehe 2.4) und Funktionen arithmetischer Typen (siehe 4.1).

Im allgemeinen besteht ein arithmetischer Ausdruck aus arithmetischen Operanden, arithmetischen Operatoren und Klammerpaaren. Er beschreibt eine Rechenvorschrift, durch welche ein Wert bestimmt wird.

Ein Operand ist zunächst jeder der genannten Spezialfälle oder ein in Klammern gesetzter arithmetischer Ausdruck.

Es gibt folgende arithmetische Operatoren, die bekannte Operationen repräsentieren:

- `**` Exponentiation
- `/` Division
- `*` Multiplikation
- `-` Subtraktion oder Multiplikation mit -1, d.h. Vorzeichenoperator
- `+` Addition oder Identität, d.h. Vorzeichenoperator

Alle diese Operatoren können binär sein, d.h. zwei Operanden verknüpfen. Sie stehen dann zwischen diesen Operanden. Als (unäre) Vorzeichenoperatoren stehen + bzw. - vor je einem Operanden. Ein

Vorzeichenoperator darf nicht unmittelbar auf einen binären Operator folgen und auch nicht auf einen vorangehenden unären Operator.

Beispiele: -Z A*B+C A*(B+C) X**(-Y)

Den Operatoren entsprechend unterscheidet man unäre und binäre Operationen.

Es ergeben sich zwei Fragen:
1. In welcher Reihenfolge werden Operationen ausgeführt, wenn mehr als ein Operator vorhanden ist? Ist beipielsweise A*B+C gleichbedeutend mit (A*B)+C oder mit A*(B+C)?
2. Wie wird verfahren, wenn Operanden von verschiedenem Typ zu verknüpfen sind?

Zu 1. Bei der Auswertung eines arithmetischen Ausdrucks wird folgende Reihenfolge eingehalten.

(1) Ausdrücke in Klammern
(2) Exponentiationen
(3) Multiplikationen und Divisionen
(4) Subtraktionen und Additionen sowie unäre Operationen

Bei ineinandergeschachtelten Klammerpaaren wird der Inhalt des innersten Klammerpaares zuerst bearbeitet. Bei Konkurrenz von Operationen gleicher Stufe wird von links nach rechts ausgewertet mit einer Ausnahme: Konkurrieren Exponentiationen miteinander, so erfolgt die Auswertung von rechts nach links. Also ist A**B**C äquivalent mit A**(B**C), A+B+C äquivalent mit (A+B)+C. Man beachte, daß wegen möglicher Rundungsfehler im allgemeinen (A+B)+C ≠ A+(B+C) ist.

Weitere Beispiele:

A/B+C ist gleichbedeutend mit (A/B)+C
-A**2 " " " -(A**2) (!)

Zu 2. Der Typ des Ergebnisses einer unären Operation ist mit dem Typ des Operanden identisch. Der Typ des Ergebnisses einer binären arithmetischen Operation ergibt sich aus der folgenden Tabelle, wo a den linken und b den rechten Operanden bezeichnet. In allen fol-

genden Tabellen wird DOUBLE als Abkürzung für DOUBLE PRECISION verwendet. Die Verknüpfung von doppeltgenauen und komplexen Operanden ist laut Norm unzulässig. Manche Compiler gestatten sie trotzdem.

a / b	INTEGER	REAL	DOUBLE	COMPLEX
INTEGER	INTEGER	REAL	DOUBLE	COMPLEX
REAL	REAL	REAL	DOUBLE	COMPLEX
DOUBLE	DOUBLE	DOUBLE	DOUBLE	-
COMPLEX	COMPLEX	COMPLEX	-	COMPLEX

Die Umwandlung einer Größe in einen anderen Typ bedeutet, daß der Wert der Größe intern dem anderen Typ entsprechend dargestellt wird. Dabei können Rundungsfehler entstehen. Wenn der Wert der umzuwandelnden Größe nicht im Bereich der darstellbaren Zahlen des anderen Typs liegt, ist keine Umwandlung möglich.

Wenn die Operanden von verschiedenem Typ sind, wird vor Ausführung der Operation der Operand, der nicht vom Typ des Ergebnisses ist, in diesen Typ umgewandelt. Eine Ausnahme bildet die Exponentiation mit Exponenten vom Typ INTEGER. Dieser wird nicht in den Typ des anderen Operanden umgewandelt.

Die Verknüpfung zweier komplexer Operanden durch zwei Sterne erzeugt als Ergebnis den Hauptwert der komplexen Potenz.

Man beachte, daß das Ergebnis der Verknüpfung zweier INTEGER-Operanden vom Typ INTEGER ist, wobei ein eventueller Divisionsrest verlorengeht. Beispielsweise ergibt 1/2 also Null, -5/2 ergibt den Wert -2. Analog ergibt 2**(-1) Null. Man beachte außerdem, daß die Auswertungsregel für jeden Teilausdruck gilt: 1/2*.5 ergibt ebenfalls Null.

Wenn in einem Ausdruck a**b die Operanden reellwertig sind und a negativ ist, dann muß b vom Typ INTEGER sein.

Ein arithmetischer Ausdruck wird als konstanter arithmetischer Aus-

druck bezeichnet, wenn außer Operatoren und Klammerpaaren nur (benannte oder unbenannte) arithmetische Konstanten darin vorhanden sind.

Für den Fall, daß das Ergebnis einer arithmetischen Operation ausserhalb des Bereichs der für die betreffende Anlage darstellbaren Zahlen liegt (Overflow) oder betragsmäßig kleiner als die kleinste darstellbare Gleitkomma-Zahl (Underflow) ist, macht die Norm keine Vorschrift. Dasselbe gilt für den Real- und Imaginärteil einer komplexen Zahl. Man muß dem jeweiligen Benutzerhandbuch entnehmen, wie diese Fälle behandelt werden. Denkbar ist beispielsweise beim Overflow ein Fehlerabbruch. Diese Behandlung ist nicht selbstverständlich. Mathematisch nicht definierte Operationen, wie Division durch Null, sind auch in FORTRAN nicht erlaubt.

2.1.7 Arithmetische Wertzuweisung

Eine mögliche Anwendung des arithmetischen Ausdrucks erfolgt in der arithmetischen Wertzuweisung (arithmetic assignment statement)

v=e

Hierin sind

v Variable oder Feldelement (siehe 2.4) eines arithmetischen Typs

e arithmetischer Ausdruck

Die arithmetische Wertzuweisung bewirkt die Auswertung von e; dann wird das Ergebnis in den Speicherplatz übertragen, der für v reserviert ist. Wenn das Ergebnis von e nicht mit dem Typ von v übereinstimmt, wird es vor der Übertragung in den Typ von v umgewandelt.

2.1.8 Arithmetisches IF

In einem Programm werden Anweisungen in der Reihenfolge ausgeführt, in der sie im Programm erscheinen, wenn nicht durch eine Steueranweisung eine Verzweigung erzwungen wird. In der

arithmetischen IF-Anweisung

IF(e)s_-,s_0,s_+

sind

e	ein reellwertiger Ausdruck
s_-,s_0, s_+	Marken, die in derselben Programmeinheit erscheinen, in der die IF-Anweisung steht

Je nachdem, ob e einen Wert kleiner Null, gleich Null oder größer Null ergibt, wird zu der durch s_-, s_0, bzw. s_+ gekennzeichneten Anweisung verzweigt. Die auf die arithmetische IF-Anweisung folgende nächste ausführbare Anweisung muß durch eine Marke gekennzeichnet sein; sonst könnte sie niemals erreicht werden.

Ist e von einem der Typen REAL oder DOUBLE PRECISION, so ist beim Test auf Null Vorsicht geboten, weil Rundungsfehler auftreten können.

2.2 Das vollständige Programm

2.2.1 PROGRAM

Jedes Hauptprogramm kann durch einen Namen gekennzeichnet werden, indem als erste Anweisung

PROGRAM pn

erscheint. Der Name pn ist global bezüglich des ausführbaren Programms, er darf daher mit anderen globalen Namen nicht identisch sein.

Globale Namen bezeichnen das Hauptprogramm, externe Prozeduren (siehe 4), benannte COMMON-Blöcke (siehe 8.1) oder BLOCK DATA-Unterprogramme (siehe 8.4).

Alle anderen Namen sind lokal, und zwar in den meisten Fällen bezüglich einer Programmeinheit, d.h. sie bezeichnen dort eindeutig bestimmte Größen wie z.B. Konstanten oder Variablen, die anderen Programmeinheiten nicht bekannt sind. Es gibt ferner Namen, die lokal bezüglich einer Anweisung (siehe 4.1.2) oder eines DO-Ele-

ments in einer DATA-Anweisung (siehe 8.3) sind.

Der globale Name pn darf nicht mit einem lokalen Namen des Hauptprogramms identisch sein. Die PROGRAM-Anweisung darf auch weggelassen werden. Namen für Hauptprogramme können für Dokumentationszwecke vergeben werden.

2.2.2 Listengesteuerte Ein-/Ausgabe

Um Programme sinnvoll arbeiten lassen zu können, werden bequem anwendbare Anweisungen für die Übertragung von Daten zum und vom Hauptspeicher benötigt. Dafür sind besonders die Anweisungen der listengesteuerten Ein-/Ausgabe (siehe 7.2)

```
READ*[,ealiste]
PRINT*[,ealiste]
```

geeignet, wo ealiste Ein-/Ausgabelisten sind.

Eine Liste ist eine Aufzählung von Listenelementen, die durch Kommas getrennt sind. In einer Ein-/Ausgabeliste können genannt werden

(1) Variablen
(2) Felder oder Feldelemente (siehe 2.4)
(3) Teilketten (siehe 5.1.3)
(4) Implizite DO-Elemente (siehe 6.2.5)

Ein in einer Ein-/Ausgabeliste erscheinender Name kann mit einem beliebigem Datentyp verbunden sein.

In einer Ausgabeliste können zusätzlich arithmetische oder auch allgemeinere Ausdrücke stehen, die gewissen einschränkenden Bedingungen genügen müssen (siehe 6.2.5). Damit können auch die bereits in Abschnitt 1.1 erwähnten Textkonstanten (siehe 5.1.2) Elemente einer Ausgabeliste sein, d.h. Texte, die in Apostrophe eingeschlossen sind. Ausgegeben wird der Text ohne die begrenzenden Apostrophe der Textkonstanten. Soll ein Apostroph ausgegeben werden, so muß entsprechend in der Textkonstanten ein Paar von Apostrophen vorhanden sein. Beispiel:

```
PRINT*,'''EIN HEIL''GER VOGEL'''
```

gibt aus

'EIN HEIL'GER VOGEL'

Durch die READ- bzw. PRINT-Anweisung wird jeweils mindestens ein (Daten-)Satz übertragen. Ein Eingabesatz kann beispielsweise aus den Zeichen in den 80 Spalten einer Lochkarte bestehen, ein Ausgabesatz aus den Zeichen, die eine Druckzeile bilden.

Ein Eingabesatz muß Konstanten in einer gültigen Form enthalten, die durch Trennzeichen voneinander abgegrenzt sind. Trennzeichen kann ein Komma, das von beliebig vielen Leerzeichen umgeben sein darf, oder ein Leerzeichen sein. Die Zuordnung zwischen einem Element der Eingabeliste einerseits und einer Konstanten in einem Eingabesatz oder in mehreren Eingabesätzen andererseits wird durch die Reihenfolge der Listenelemente und der Konstanten bestimmt.

Für die Eingabe gültige Formen arithmetischer Konstanten sind die in Abschnitt 2.1.1 erklärten mit der Einschränkung, daß keine Leerzeichen eingestreut sein dürfen, weil diese Trennungsfunktion haben.

Die Eingabekonstanten müssen im Typ mit den zugehörigen Listenelementen übereinstimmen, sonst erfolgt bei der Ausführung des Programms ein Fehlerabbruch. Ausnahme: Ganzzahlige Gleitkomma-Konstanten können auch ohne Dezimalpunkt angegeben werden.

Das Erreichen eines Satzendes, z.B. des Endes einer Eingabelochkarte, wirkt in den meisten Fällen so, als ob ein Leerzeichen als Trennzeichen vorhanden wäre. Ausnahmen siehe Abschnitt 7.2.

Die listengesteuerte Ausgabe erfolgt in einer compilerabhängigen Form.

Jede READ- und jede PRINT-Anweisung überträgt mindestens einen neuen Satz. Ist ein Satz übertragen und es sind noch in der Ein-/Ausgabeliste Elemente übrig, so wird ein weiterer Satz übertragen usw., bis die Ein-/Ausgabeliste abgearbeitet ist.

Durch READ* ohne Eingabeliste wird ein Eingabesatz nicht gelesen, d.h. übergangen, durch PRINT* ohne Ausgabeliste wird eine Leerzeile gedruckt.

Die listengesteuerte Ein-/Ausgabe ist eine zweckmäßige Erweiterung älterer FORTRAN-Versionen. Sie steht allerdings im FORTRAN-Subset nicht zur Verfügung (siehe Anhang C).

2.2.3 END, STOP, PAUSE

Die Anweisung

END

wurde schon im Abschnitt 1.1 erwähnt. Sie muß die letzte Anweisung jeder Programmeinheit sein und kommt nirgendwo sonst vor. Wird sie in einem Hauptprogramm ausgeführt, so wird das ausführbare Programm insgesamt beendet. Wird sie in einem Funktions- oder Subroutine-Unterprogramm (siehe 4.1.3, 4.2) ausgeführt, so wird in die rufende Programmeinheit zurückverzweigt.

Die Anweisungen STOP und PAUSE werden in dem Programm-Beispiel dieses Abschnitts nicht benötigt und hier nur der Vollständigkeit halber angegeben.

Die Ausführung der Anweisung

STOP n

beendet ein ausführbares Programm. Dabei ist n eine Kette von höchstens 5 Ziffern oder eine Textkonstante. Die Verwendung von n ist anlagenabhängig; beispielsweise kann n ins Ausgabeprotokoll gedruckt werden.

Die Norm enthält ferner die Anweisung

PAUSE n

Sie unterbricht die Ausführung eines Programms. Über die Bedeutung von n ist wie bei der STOP-Anweisung nichts Bestimmtes vorgeschrieben. Durch PAUSE kann beispielsweise einem Operateur die Anweisung zum Auflegen eines Magnetbandes gegeben werden. Dieser muß dann auch die Fortsetzung des Programms veranlassen, was durch das Programm selbst nicht mehr geschehen kann.

2.2.4 Ein FORTRAN-Programm als Anwendungsbeispiel

Die folgende Aufgabe ermöglicht die Anwendung der meisten der oben erklärten Anweisungen: Es ist ein Polynom $P = a_3x^3+a_2x^2+a_1x+a_0$ in einem Intervall [a,b] mit einer Schrittweite h=(b-a)/m zu tabellieren.

Weil häufig benötigte Funktionen oft durch Polynome approximiert werden, ist eine Rechenvorschrift, die ein Polynom mit minimalem Aufwand auswertet, d.h. mit so wenig arithmetischen Operationen wie möglich, von großer Bedeutung. Es kommt das Hornersche Schema $P = ((a_3x+a_2)x+a_1)x+a_0$ in Frage. Bemühungen, etwas Besseres zu finden, waren übrigens praktisch erfolglos.

An dem Beispiel soll gleichzeitig das Auftreten von Rundungsfehlern demonstriert werden. Die Koeffizienten a_3, a_2, a_1, a_0 werden daher so gewählt, daß das Polynom in x=1, x=2, x=3 Nullstellen hat. Daher wird $P = (x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6$ gesetzt. In einer PARAMETER-Anweisung werden die Konstanten A3, A2, A1, A0 erklärt. Werte für die Intervallgrenzen a, b und für die Zahl m werden eingelesen. Damit kann das Programm lauten

```
      PROGRAM POLYNO
      PARAMETER(A3=1.,A2=-6.,A1=11.,A0=-6.)
      READ*,A,B,M
      H=(B-A)/M
      X=A
10    P=((A3*X+A2)*X+A1)*X+A0
      PRINT*,X,P
      X=X+H
      M=M-1
      IF(M)11,10,10
11    END
```

Mit dem Eingabesatz

```
0.,4.,40
```

erzeugte eine Anlage UNIVAC 1100/83, die 27 Bit Mantissenlänge besitzt, folgende Ausgabe.

.00000000	-6.0000000
.10000000+000	-4.9590001
.20000000	-4.0320000
.30000000	-3.2130001
.39999999	-2.4960001
.49999999	-1.8750001
.59999999	-1.3440000
.69999999	-.89700013
.79999997	-.52800006
.89999997	-.23100013
.99999996	.00000000
1.0999999	.17099994
1.1999999	.28799999
1.2999999	.35699993
1.3999999	.38400006
1.4999999	.37500000
1.5999999	.33600008
1.6999999	.27299994
1.7999999	.19200021
1.8999999	.98999977-001
1.9999999	.11920929-006
2.0999999	-.98999977-001
2.1999999	-.19199991
2.2999999	-.27300006
2.3999999	-.33599979
2.4999999	-.37500006
2.5999998	-.38399982
2.6999998	-.35700029
2.7999998	-.28800017
2.8999998	-.17100042
2.9999998	-.59604645-007
3.0999998	.23099917
3.1999998	.52799946
3.2999998	.89699888
3.3999998	1.3439990
3.4999998	1.8749987
3.5999998	2.4959986
3.6999998	3.2129982
3.7999998	4.0319984
3.8999998	4.9589972
3.9999998	5.9999979

Es fällt auf, daß die Werte von X durch Rundungsfehler verfälscht sind. Damit sind auch die Funktionswerte mit Fehlern behaftet. Das wird in der Tabelle besonders deutlich an den Stellen x=2 und x=3, für die von Null verschiedene Funktionswerte ausgegeben werden.

Der Grund besteht darin, daß das Programm zwar logisch richtig, aber hinsichtlich der Rundungsfehlersituationen ungünstig ist. Aus den Eingabewerten wird eine Schrittgröße errechnet, die nicht exakt

.1, sondern rundungsfehlerbedingt etwas kleiner ist. Dieser Rundungsfehler wird in der Anweisung X=X+H akkumuliert.

Man kann sich bemühen, ungünstige Rundungsfehlersituationen zu vermeiden, beispielsweise indem man das Programm abändert in

```
      PARAMETER(A3=1.,A2=-6.,A1=11.,A0=-6.)
      READ*,A,B,M
      N=M
      X=A
10    P=((A3*X+A2)*X+A1)*X+A0
      PRINT*,X,P
      M=M-1
      X=(A*M+(N-M)*B)/N
      IF(M)11,10,10
11    END
```

Dieselbe Anlage U1100/83 erzeugt damit die Ausgabe

```
 .00000000      -6.0000000
 .10000000+000  -4.9590001
 .20000000      -4.0320000
 .30000000      -3.2130001
 .40000000      -2.4960000
 .50000000      -1.8750000
 .59999999      -1.3440000
 .70000000      -.89700001
 .80000000      -.52800000
 .90000000      -.23099995
 1.0000000       .00000000
 1.1000000       .17099994
 1.2000000       .28800005
 1.3000000       .35699993
 1.4000000       .38400000
 1.5000000       .37500000
 1.6000000       .33600003
 1.7000000       .27300000
 1.8000000       .19199985
 1.9000000       .99000037-001
 2.0000000       .00000000
 2.1000000      -.98999918-001
 2.2000000      -.19199985
 2.3000000      -.27300006
 2.4000000      -.33599979
 2.5000000      -.37500000
 2.6000000      -.38400012
 2.7000000      -.35699999
 2.8000000      -.28800017
```

```
2.9000000      -.17099977
3.0000000       .00000000
3.1000000       .23099995
3.2000000       .52800018
3.3000000       .89699966
3.4000000      1.3440002
3.5000000      1.8750000
3.6000000      2.4959996
3.7000000      3.2129999
3.8000000      4.0319993
3.9000000      4.9599991
4.0000000      6.0000000
```

Es sind fast keine Rundungsfehler mehr zu bemerken. Allerdings verursacht das Programm zur Berechnung der Variablen X höheren Rechenaufwand.

2.3 GOTO

Die GOTO-Anweisung, von der es drei Formen gibt, ist wie die arithmetische IF-Anweisung eine Steueranweisung.

(1) Die unbedingte GOTO-Anweisung lautet

```
GOTO s
```

wo s eine Marke in derselben Programmeinheit ist, in der die GOTO-Anweisung steht. Das Programm wird mit der Anweisung, die durch s gekennzeichnet ist, fortgesetzt.

(2) Bei der berechneten GOTO-Anweisung kann unter mehreren Sprungzielen ausgewählt werden:

```
GOTO(s[,s]...)[,]i
```

Es sind

s Marken. Dieselbe Marke kann mehrmals in derselben berechneten GOTO-Anweisung stehen

i INTEGER-Ausdruck

Die Marken werden mit 1 beginnend von links gezählt. Über den Wert von i kann die Marke angewählt werden, bei der das Programm fortgesetzt wird. Zeigt i nicht auf eine Marke, so bleibt die berechnete GOTO-Anweisung wirkungslos, d.h. es wird mit der ihr folgenden aus-

führbaren Anweisung das Programm fortgesetzt.

(3) Die zugeordnete GOTO-Anweisung ermöglicht, ähnlich wie die berechnete GOTO-Anweisung, eine Auswahl zwischen verschiedenen Sprungzielen.

Der Ausführung einer zugeordneten GOTO-Anweisung muß in derselben Programmeinheit eine Anweisung

ASSIGN s TO i

vorausgegangen sein, wo

s Marke
i INTEGER-Variable

ist. Die Marke s wird der Variablen i zugeordnet. Das darf nicht mit einer Wertzuweisung verwechselt werden, die bekanntlich die Form v=e hat.

Die zugeordnete GOTO-Anweisung lautet schließlich

GOTO i [[,](s[,s]...)]

Hierin sind

i Name einer INTEGER-Variablen, der zuletzt vorher in einer ASSIGN-Anweisung eine Marke zugewiesen wurde
s Marken derselben Programmeinheit

Es wird mit derjenigen Anweisung fortgefahren, deren Marke in einer ASSIGN-Anweisung der Variablen mit dem Namen i zugeordnet wurde. Wenn eine Liste von Marken vorhanden ist, muß diese Marke darin enthalten sein. Wenn die Liste fehlt, ergänzt der Compiler eine Liste aller Marken der Programmeinheit.

Eine Anwendungsmöglichkeit für die zugeordnete GOTO-Anweisung wird durch die Aufgabe eröffnet, in einer Programmeinheit von einer beliebigen Stelle aus zu einer Folge von Anweisungen zu verzweigen und nach deren Ausführung mit der auf die Verzweigung folgenden ausführbaren Anweisung fortzufahren.

Die wiederholt zu durchlaufenden Anweisungen bilden ein sogenanntes inneres Unterprogramm, das in der Programmeinheit aufgerufen wird, in der es enthalten ist. Beispiel: Bei Anweisung 100 beginnt ein

Unterprogramm für die Lösung von zwei Gleichungen mit zwei Unbekannten x, y

$$a \cdot x + b \cdot y = c$$
$$d \cdot x + e \cdot y = f$$

Eingabedaten sind Konstanten für a, b, c, d, e, f, für die vorausgesetzt wird, daß a e - d b $\neq$ 0 gilt und daß das Gleichungssystem mit den Konstanten gut konditioniert ist. Dann kann die eindeutige Lösung bis auf Rundungsfehler berechnet werden. Das Unterprogramm greift auf feste Speicherplätze für a, b, c, d, e, f zu.

```
      ASSIGN 1 TO I
      GOTO 100
1     PRINT*,'LOESUNG1: X=',X,' Y=',Y
      :
100   DET=A*E-D*B
      X=(C*E-F*D)/D
      Y=(A*F-D*C)/D
      GOTO I
      :
      (Verändere die Variablen A, B, C, D, E, F und führ
      das Programm an einer anderen Stelle nochmals aus:)
      ASSIGN 2 TO I
      GOTO 100
2     PRINT*,'LOESUNG2: X=',X,' Y=',Y
      :
      END
```

2.4 Felder

Dieser Abschnitt erfaßt die arithmetischen Datentypen und auch den Typ LOGICAL (siehe 3). Für Felder des Typs CHARACTER gibt es Besonderheiten (siehe 5.1.1).

Ein Feld ist eine nichtleere geordnete Menge von Daten, gekennzeichnet durch den Namen eines Feldes. Es besteht aus Feldelemen-

ten.

Beispielsweise können die Elemente von Matrizen und Vektoren zu Feldern zusammengefaßt werden.

Ein Feldelement ist durch einen Index oder durch mehrere Indizes innerhalb eines Feldes eindeutig gekennzeichnet.

Die Reservierung von Speicherplatz für ein Feld erfolgt, indem eine Felddefinition

a(d[,d]...)

in einer der folgenden Spezifikationsanweisungen erscheint: Typanweisung, DIMENSION-Anweisung (siehe unten) oder COMMON-Anweisung (siehe 8.1). In der Felddefinition sind

a Name eines Feldes

d Indexbereich, der durch die untere und die obere Grenze eines Index in der Form [d1:]d2 erklärt ist, wo d1 die untere, d2 die obere Indexgrenze bedeutet. Fehlt d1:, so wird die untere Indexgrenze auf 1 gesetzt. Bis auf weiteres wird verlangt, daß d1 und d2 konstante INTEGER-Ausdrücke sind. Verallgemeinerungen dieser Regel werden in Abschnitt 4.3.2 angegeben. Die Bedingung d1 $\leq$ d2 muß erfüllt sein.

Die Vielfachheit der Indizierung eines Feldes wird auch als Anzahl der Dimensionen eines Feldes (kurz: Dimension eines Feldes) bezeichnet. Sie kann höchstens 7 sein. Die Größe einer Dimension ist der Wert d2 - d1 + 1.

Die Größe eines Feldes ist die Anzahl seiner Feldelemente.

Die DIMENSION-Anweisung lautet

DIMENSION a(d,[d]...)[,a(d[,d]...)]...

Beispiele:

(1) DIMENSION A(10,20),X(10),B(10)

reserviert 10*20=200 Speicherplätze für das Feld A, je 10 Speicherplätze für die Felder X und B. Mit der Namensregel sind die Felder vom Typ REAL.

(2)
```
COMPLEX C
DIMENSION C(-10:10)
```

reserviert 21 Speicherplätze für das Feld C, das den Typ COMPLEX hat. Diesen beiden Anweisungen äquivalent ist

(3)
```
COMPLEX C(-10:10)
```

Der Name eines Feldelements hat die Form

```
a(i[,i]...)
```

Hierin sind

a Name eines Feldes

i Ausdrücke vom Typ INTEGER, genannt Indexausdrücke

Die Namen eines Feldes und eines Feldelements werden im folgenden meistens kurz als Feld und Feldelement bezeichnet. Die Anzahl der Indexausdrücke eines Feldelements muß gleich der Dimension des Feldes sein.

Mit Feldelementen kann man im Programm wie mit Variablen des gleichen Typs arbeiten.

Beispiel:

```
DIMENSION A(10,10),X(10),B(10)
...
N=10
READ*,B(N),A(N,N)
X(N)=B(N)/A(N,N)
...
```

Ein Feld kann auch eingelesen oder ausgegeben werden, indem sein Name in einer Ein-/Ausgabeliste ohne Indizes genannt wird. Es werden dann alle Elemente des Feldes in der Weise angesprochen, daß der erste Index beginnend mit der unteren Indexgrenze vollständig durchvariiert wird, wobei alle anderen Indizes auf ihre untere Indexgrenze gesetzt sind. Danach wird der zweite Index um 1 erhöht, der erste erneut durchvariiert usw., bis alle Indizes ihre oberen Indexgrenzen erreicht haben. Die Elemente einer Matrix werden auf diese Weise also spaltenweise angesprochen.

In der angegebenen Reihenfolge belegen die Feldelemente auch auf-

einanderfolgende Worte des Hauptspeichers. Die Vorschrift, die für die Feldelemente eine lineare (in der FORTRAN-Sprechweise eindimensionale) Anordnung im Speicher vermittelt, wird häufig als Speicherabbildungsfunktion bezeichnet.

Sei die Reihenfolge der Feldelemente im Speicher durch die Platznummern 1, 2, 3,... gekennzeichnet, seien $i_1,\ldots,i_j$ die Indizes eines Feldelements, $c_1,\ldots,c_j$ die unteren, $d_1,\ldots,d_j$ die oberen Indexgrenzen des Feldes mit $j \leq 7$, $l_k = d_k - c_k + 1$ die Größe der k-ten Dimension für k=1,...,j-1, so lautet die Speicherabbildungsfunktion in der Form des Hornerschen Schemas geschrieben

$$s(i_1,\ldots,i_j) = 1 + i_1-c_1+l_1*(i_2-c_2+l_2*(\ldots+l_{j-2}*(i_{j-1}-c_{j-1}+l_{j-1}*(i_j-c_j)\ldots)$$

Man beachte, daß die Speicherabbildungsfunktion von der oberen Indexgrenze des letzten Index, d_j, unabhängig ist. Nach der Abbildungsvorschrift bewirkt die Anweisung

```
DIMENSION A(3,4,5)
```

also, daß die Feldelemente in folgender Reihenfolge im Speicher stehen: A(1,1,1), A(2,1,1), A(3,1,1), A(1,2,1), A(2,2,1) ,..., A(2,4,5) A(3,4,5). Die Eingabe kann erfolgen durch

```
READ*,A
```

Die Festlegung der Reihenfolge, in der die Konstanten der Feldelemente angegeben werden müssen, und die Tatsache, daß alle Elemente angesprochen werden, eine Auswahl von Teilmengen also nicht möglich ist, schränkt die Anwendbarkeit dieser Form der Ein-/Ausgabe ein. Implizite DO-Elemente beheben diesen Mangel (siehe 6.2.5).

Oft ist es zweckmäßig, für die Indexgrenzen benannte Konstanten anzugeben.

Beispiel: Die Dimensionierung eines zweidimensionalen Feldes (a_{ij}), i=1,...,6; j=1,...,7 kann durch

```
PARAMETER (M=6,N=7)
DIMENSION A(M,N)
```

erfolgen statt durch

```
DIMENSION A(6,7)
```

Wenn im Programm die Indexgrenzen M, N geändert werden müssen, genügt es, die Änderung in der PARAMETER-Anweisung vorzunehmen. Allerdings kann nach dieser Änderung die Neuübersetzung des FORTRAN-Programms nicht vermieden werden. In FORTRAN ist nämlich die dynamische Feldvereinbarung unzulässig, bei der die Speicherplatzverteilung während der Ausführung eines Programms erfolgt. Deshalb dürfen Variablen nicht als Feldgrenzen angegeben werden.

2.5 DO

Die DO-Anweisung ermöglicht es, eine Folge von Anweisungen, die Wiederholungsbereich genannt wird, beliebig oft zu durchlaufen. Vor jedem neuen Durchlauf wird ein Parameter, die Laufvariable, geändert. Die DO-Anweisung hat die Form

DO s [,] i=e_1,e_2[,e_3]

Hierin sind

s	Marke der letzten Anweisung im Wiederholungsbereich
i	reellwertige Laufvariable
e_1, e_2, e_3	reellwertige Ausdrücke. Sie bezeichnen in dieser Reihenfolge Anfangswert, Endwert und Schrittweite der Laufvariablen. Fehlt e_3, so ist die Schrittweite +1

Die Anwendung der DO-Anweisung kann an der Aufgabe erläutert werden, die Euklidische Norm eines Vektors zu berechnen.

```
      PARAMETER (N=...)
      REAL NORM,A(N)
      READ*,A
      SUMME=0.
      DO 10 I=1,N
10    SUMME=SUMME+A(I)**2
      NORM=SUMME**.5
      PRINT*,NORM
      END
```

Die Wirkung der DO-Anweisung zusammen mit ihrem Wiederholungsbereich kann man durch das folgende Flußdiagramm beschreiben.

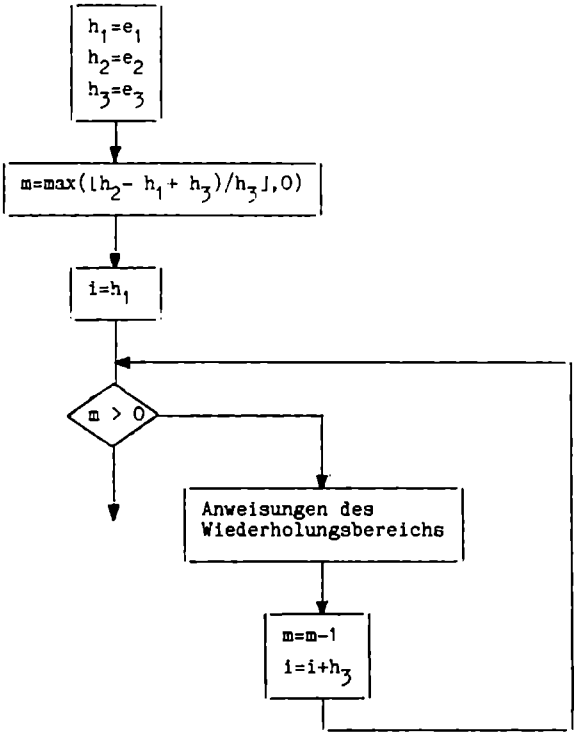

Bild 1

Der Wiederholungsbereich zusammen mit der DO-Anweisung heißt DO-Schleife. Ihre Bearbeitung wird von dem Durchlaufzähler m gesteuert. Im Unterschied zu vielen alten FORTRAN-Versionen wird der Wiederholungsbereich keinmal durchlaufen, wenn m anfänglich den Wert Null hat.

Die Laufvariable i darf nicht im Wiederholungsbereich abgeändert werden. Vor der erstmaligen Ausführung des Wiederholungsbereichs

werden die Ausdrücke e_1, e_2 und ggf. e_3 ausgewertet, in den Typ von i umgewandelt und in h_1, h_2, h_3 gespeichert. Im Wiederholungsbereich können daher die Werte von e_1, e_2, e_3 nicht geändert werden.

Die letzte Anweisung des Wiederholungsbereichs darf weder eine GOTO-, arithmetische IF-, STOP-, END- oder DO-Anweisung sein, noch eine der Anweisungen Block IF, ELSE IF, ELSE oder END IF (siehe 3.5) und auch keine RETURN-Anweisung (siehe 4.4). Eine logische IF-Anweisung (siehe 3.4) ist zulässig.

Es ist klar, daß nicht unter Umgehung der DO-Anweisung in den zugehörigen Wiederholungsbereich gesprungen werden darf, da dann die Laufvariable nicht definiert wäre.

Wenn e_1, e_2, e_3 nicht vom Typ INTEGER sind, kann es geschehen, daß infolge von Rundungsfehlern der letzte Durchlauf des Wiederholungsbereichs unterbleibt. Wenn die genannten Parameter mit einem der Typen REAL oder DOUBLE PRECISION vereinbart werden, ist daher Vorsicht geboten.

Ein Wiederholungsbereich kann seinerseits DO-Anweisungen enthalten, so daß eine Schachtelung von DO-Schleifen entsteht. Die Wiederholungsbereiche geschachtelter DO-Anweisungen können eine gemeinsame letzte ausführbare Anweisung besitzen. Die letzte Anweisung eines inneren Wiederholungsbereichs darf nicht hinter der letzten Anweisung eines ihn umfassenden äußeren Wiederholungsbereichs stehen. Wird zur gemeinsamen letzten Anweisung der Wiederholungsbereiche geschachtelter DO-Schleifen verzweigt, so wird dies als Verzweigung zum Ende des innersten Wiederholungsbereichs aufgefaßt.

Es gibt compilerabhängige Schranken für die Schachtelungstiefe. Die Norm macht hierzu keine Vorschriften.

Ein Wiederholungsbereich kann, bevor er vollständig abgearbeitet ist, durch eine Verzweigung verlassen werden. Die Laufvariable bleibt dann definiert, wie es dem Flußdiagramm Bild 1 zu entnehmen ist.

Nach normaler Beendigung der DO-Anweisung behält die Laufvariable ebenfalls den zuletzt zugewiesenen Wert, so wie es in Bild 1 dar-

gestellt ist. Man beachte, daß das nicht der Wert ist, den sie beim letzten Durchlauf des Wiederholungsbereichs hatte.

Innerhalb eines Wiederholungsbereichs dürfen Unterprogramme (siehe 4) aufgerufen werden. Sonst ist es unzulässig, einen Wiederholungsbereich zu verlassen und später in diesen zurückzukehren.

Man kann nun eine n*n-Matrix folgendermaßen einlesen. Es sei $n \leq 20$ vorausgesetzt.

```
      PARAMETER (NP=20)
      DIMENSION A(NP,NP)
      READ*,N
      DO 10 I=1,N
      DO 10 J=1,N
10    READ*,A(I,J)
```

Der Nachteil der Methode ist, daß mit jedem READ ein neuer Satz angefordert wird. Beim Arbeiten mit Lochkarten muß daher der Wert für jedes Feldelement auf je einer Lochkarte stehen. Abhilfe schaffen die schon erwähnten impliziten DO-Elemente (siehe 6.2.5).

Die DO-Anweisung wird nun in einem Programmausschnitt angewendet, der die Lösung von sogenannten gestaffelten Gleichungssystemen

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{ii}x_i + \cdots + a_{in}x_n &= b_i \\ &\cdots \\ a_{nn}x_n &= b_n \end{aligned}$$

errechnet, wo $a_{ii} \neq 0$ für i=1,2,...,n und $n \geq 2$ vorausgesetzt wird, das heißt, die Gleichungssysteme haben je eine eindeutige Lösung. Diese wird nach dem in dem Flußdiagrammausschnitt Bild 2 angegebenen Verfahren ermittelt.

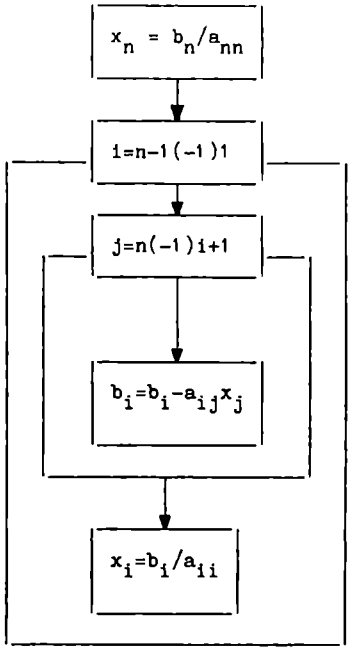

Bild 2

Entsprechende FORTRAN-Anweisungen können dann lauten

```
      X(N)=B(N)/A(N,N)
      DO 10 I=N-1,1,-1
      DO 11 J=N,I+1,-1
11    B(I)=B(I)-A(I,J)*X(J)
10    X(I)=B(I)/A(I,I)
```

Es ist nicht erforderlich, daß die Laufvariable im Wiederholungsbereich erscheint. Sie kann ausschließlich als Zähler verwendet werden. Im Programmbeispiel POLYNO (siehe 2.2.4) kann man wie folgt umformulieren:

```
      ...
      DO 10 I=M,0,-1
      P=((A3*X+A2)*X+A1)*X+A0
      PRINT*,X,P
10    X=X+H
      END
```

2.6 CONTINUE

Besonders in Verbindung mit der DO-Anweisung spielt die leere Anweisung

```
      CONTINUE
```

eine Rolle. Sie ist ausführbar und kann daher markiert werden; sie bewirkt aber nichts. Indem man mit CONTINUE als letzter Anweisung einen Wiederholungsbereich abschließt, kann man faktisch das Verbot überwinden, daß dort keine GOTO-, arithmetische IF-Anweisung usw. stehen darf.

Eine weitere Anwendungsmöglichkeit der CONTINUE-Anweisung ergibt sich, wenn unter einer Bedingung der Wiederholungsbereich nicht vollständig abzuarbeiten ist. Beispiel: Zähl die positiven Komponenten eines Vektors.

```
      PARAMETER (N=...)
      DIMENSION X(N)
      ...
      J=0
      DO 10 I=1,N
      IF(X(I))10,10,11
11    J=J+1
10    CONTINUE
      ...
```

Man beachte: Wenn irgendwo im Wiederholungsbereich festgestellt wird, daß ein Durchlauf zu beenden und ggf. ein weiterer zu initiieren ist, so muß zur letzten Anweisung des Wiederholungsbereichs verzweigt werden und nicht zur zugehörigen DO-Anweisung.

3 Arbeiten mit logischen Daten

Die beiden Wahrheitswerte "falsch", "wahr" werden durch die logischen Konstanten .FALSE., .TRUE. dargestellt. Die interne Darstellung eines Wahrheitswertes erfolgt nach der Norm in einem Wort, aus dem aber nur ein Bit gebraucht wird. Gewöhnlich wird das in einem Wort ganz rechts stehende Bit genommen, wobei 0 den Wert "falsch", 1 den Wert "wahr" bedeutet.

Logische Größen werden mit der IMPLICIT-Anweisung oder mit der logischen Typanweisung

```
LOGICAL v[,v]...
```

erklärt (siehe auch 2.1.5).

3.1 Vergleichsausdrücke

Es gibt 6 Vergleichsoperatoren, die in der folgenden Tabelle zusammengestellt sind.

Vergleichs-operator	gebräuchliches Symbol	Bedeutung
.GT.	>	greater than
.GE.	≥	greater or equal
.LT.	<	less than
.LE.	≤	less or equal
.EQ.	=	equal
.NE.	≠	not equal

Mit den Vergleichsoperatoren werden Vergleichsausdrücke e_1 op e_2 gebildet, deren Auswertung einen logischen Wert ergibt. Darin ist op Vergleichsoperator, e_1 , e_2 sind entweder beide arithmetische oder Textausdrücke (siehe 5.2.1).

In einem Vergleichsausdruck darf jeder der Operanden e_1 , e_2 reellwertig sein. Komplexe Operanden sind nur zusammen mit den Vergleichsoperatoren .EQ. oder .NE. zugelassen. Wenn die einem Vergleichsausdruck entsprechende Relation erfüllt ist, ist sein Wert "wahr", sonst "falsch".

3.2 Logischer Ausdruck

Es gibt die fünf logischen Operatoren .NOT., .AND., .OR., .EQV., .NEQV., für die sonst häufig die Zeichen $\neg$, $\wedge$, $\vee$, $\equiv$, $\not\equiv$ gebraucht werden. Repräsentieren a, b logische Werte und wird für den Wert "wahr" abkürzend T, für "falsch" F geschrieben, so können die durch die logischen Operatoren vermittelten Abbildungen durch die folgenden Tabellen erklärt werden:

$f_\neg$:

a	a
T	F
F	T

$f_\wedge$:

a \ b	T	F
T	T	F
F	F	F

$f_\vee$:

a \ b	T	F
T	T	T
F	T	F

$f_\equiv$:

a \ b	T	F
T	T	F
F	F	T

$f_{\not\equiv}$:

a \ b	T	F
T	F	T
F	T	F

Es handelt sich also um eine unäre und vier binäre Operationen (Funktionen, Abbildungen). Sie heißen Negation, Konjunktion, Disjunktion, Äquivalenz und Antivalenz. In der Aussagenlogik wird gezeigt, daß jede logische Operation $f : \{T,F\}^2 \to \{T,F\}$ durch die drei Operationen $f_\neg$, $f_\wedge$, $f_\vee$ darstellbar ist. Im nächsten Programmbeispiel wird von dieser Tatsache Gebrauch gemacht werden.

Bei den logischen Operationen gibt es analog zu den arithmetischen Vorrangregeln, die besagen, daß bei der Auswertung logischer Ausdrücke folgende Reihenfolge eingehalten wird:

(1) Negationen
(2) Konjunktionen
(3) Disjunktionen
(4) Äquivalenzen und Antivalenzen

In üblicher Weise können diese Regeln durch das Setzen von Klammerpaaren aufgehoben werden. Es ist klar, daß vor den logischen Ausdrücken die Vergleichsausdrücke, vor diesen arithmetische Ausdrücke ausgewertet werden.

Ein logischer Ausdruck wird nach folgenden Regeln gebildet:

(1) Eine logische Konstante, eine benannte logische Konstante, eine logische Variable, ein logisches Feldelement, eine logische Funktion (siehe 4.1) oder ein Vergleichsausdruck ist ein logischer Ausdruck.

(2) e_1 op e_2 , wo e_1, e_2 logische Ausdrücke sind und op logischer Operator .AND., .OR., .EQV. oder .NEQV., ist logischer Ausdruck.

(3) .NOT.e, wo e logischer Ausdruck ist, ist logischer Ausdruck, wenn e nicht mit .NOT. beginnt.

(4) (e), wo e logischer Ausdruck ist, ist logischer Ausdruck.

3.3 Logische Wertzuweisung

Die logische Wertzuweisung (logical assignment statement) lautet

```
v=e
```

Hierin sind

v	logische Variable
e	logischer Ausdruck

3.4 Logische IF-Anweisung

Durch die logische IF-Anweisung wird die Ausführung einer einzelnen ausführbaren Anweisung vom Wert eines logischen Ausdrucks abhängig gemacht. Sie hat die Form

```
IF(e)st
```

Hierin sind

e	logischer Ausdruck
st	ausführbare Anweisung, aber weder eine weitere logische IF-

Anweisung, noch eine DO- oder eine END-Anweisung und auch keine der Anweisungen Block IF, ELSE IF, ELSE oder END IF (siehe 3.5)

Wenn e den Wert "wahr" ergibt, wird die Anweisung st ausgeführt. Wenn e den Wert "falsch" ergibt, wird st ignoriert und mit der auf die logische IF-Anweisung folgenden ausführbaren Anweisung fortgefahren.

3.5 Block IF, END IF, ELSE IF, ELSE

Die Block IF genannte Anweisung

```
IF(e)THEN
```

wo e ein logischer Ausdruck ist und die Anweisung

```
END IF
```

ermöglichen es, die Ausführung einer Folge von Anweisungen, die den IF-Block bilden, vom Wert eines logischen Ausdrucks abhängig zu machen. Es sind dazu Konstruktionen der Form

```
IF(e)THEN
   :
   (Anweisungen des IF-Blocks)
   :
END IF
```

erforderlich.

Vor der Erklärung weiterer Einzelheiten werden diese Anweisungen und die logische IF-Anweisung in einem Programm zu folgender Aufgabe aus dem Buch "99 Logeleien von Zweistein" angewendet: "Meiers werden uns heute abend besuchen", kündigt Herr Müller an. "Die ganze Familie, also Herr und Frau Meier nebst ihren drei Söhnen Tim, Kai und Uwe?" fragt Frau Müller bestürzt. Darauf Herr Müller, der keine Gelegenheit vorübergehen läßt, seine Frau zum logischen Denken anzureizen: "Nein, ich will es dir so erklären: Wenn Vater Meier kommt, dann bringt er seine Frau mit. Mindestens einer der

beiden Söhne Uwe oder Kai kommt. Entweder kommt Frau Meier oder Tim. Entweder kommen Tim und Kai oder beide nicht. Und wenn Uwe kommt, dann auch Kai und Herr Meier. So, jetzt weißt du, wer uns heute abend besuchen wird."

Die Aussage "entweder a oder b" entspricht der Antivalenz, die Aussage "entweder a und b oder weder a noch b" der Äquivalenz. Die Aufgabe erfordert aber auch eine Operation, die nicht unmittelbar von FORTRAN zur Verfügung gestellt wird, nämlich die Implikation "aus a folgt b", in Zeichen $a \Rightarrow b$. Sie lautet in Tabellenform

f :

a \ b	T	F
T	T	F
F	T	T

Eine äquivalente Definition der Implikation mit einem logischen Ausdruck ist

$$f_{\Rightarrow}(a,b) = \neg a \vee b$$

Das Programm zu der Aufgabe führt für Meier, Frau Meier, Tim, Kai und Uwe 5 logische Variablen ein, besetzt diese mit allen Variationen der Wahrheitswerte "wahr" und "falsch" und testet, unter welchen Variationen Meiers Aussagen "wahr" sind. Dabei ergibt sich, ob es eine oder keine Lösung oder mehrere Lösungen gibt. In diesem Fall ist es eine. Tim und Kai sind zu erwarten.

```
      PROGRAM LOGIK
C     S IST EINE LOGISCHE HILFSVARIABLE, DIE DEN
C     WERT 'WAHR' ERHAELT, SOBALD EINE LOESUNG
C     GEFUNDEN IST
      LOGICAL HMEIER,FMEIER,TIM,KAY,UWE,S
      S=.FALSE.
      DO 10 I=0,1
      HMEIER=I.EQ.1
      DO 10 J=0,1
      FMEIER=J.EQ.1
```

```
      DO 10 K=0,1
      TIM=K.EQ.1
      DO 10 L=0,1
      KAY=L.EQ.1
      DO 10 M=0,1
      UWE=M.EQ.1
      IF((.NOT.HMEIER.OR.FMEIER).AND.(UWE.OR.KAY).AND.
     *   J.NE.K.AND.K.EQ.L.AND.(.NOT.UWE.OR.KAY.AND.
     *   HMEIER))THEN
C     HIER WURDEN DIE LAUFVARIABLEN J, K, L BENUTZT, UM
C     DIE ANTIVALENZ DURCH J.NE.K UND DIE AEQUIVALENZ
C     DURCH K.EQ.L ZU BILDEN
         PRINT*,'ES SIND ZU ERWARTEN'
         IF(HMEIER)PRINT*,'HERR MEIER'
         IF(FMEIER)PRINT*,'FRAU MEIER'
         IF(TIM)PRINT*,'TIM'
         IF(KAY)PRINT*,'KAY'
         IF(UWE)PRINT*,'UWE'
         S=.TRUE.
      END IF
10    CONTINUE
      IF(.NOT.S)PRINT*,'KEINE LOESUNG GEFUNDEN'
      END
```

Das Programm druckt also eine Lösung aus, wenn der logische Ausdruck "wahr" ist, in den die genannten Bedingungen eingehen. Die Hilfsvariable S wird verwendet, um ggf. auf einen Widerspruch in Meiers Aussagen durch Ausgabe des Textes KEINE LOESUNG GEFUNDEN hinzuweisen.

Man kommt zu einer eleganteren Lösung der Aufgaben dieses Typs für n Variablen, wenn man die Werte "falsch", "wahr" in den 2^n Variationen durch die Werte 0, 1 der Dualstellen einer Dualzahl $b_1 \ldots b_n$ ersetzt denkt. (In der oben angegebenen Lösung werden die Dualstellen durch die Laufvariablen I,...,M repräsentiert). Die Variablen HMEIER,...,UWE sind nun durch einen logischen Vektor zu ersetzen und man kann schreiben

```
      LOGICAL V(5),S
      S=.FALSE.
      DO 10 I=1,32
      K=I
      DO 11 J=1,5
      V(J)=MOD(K,2).EQ.1
C     MOD BEZEICHNET EINE INNERE STANDARDFUNKTION (SIEHE
C     4.1.1). ES WURDE DARAUF VERZICHTET, DIE DURCH
C     MOD(K,2) ERRECHNETEN DUALSTELLEN EINEM INTEGER-FELD
      ZU SPEICHERN
11    K=K/2
      IF((.NOT.V(1).OR.V(2)).AND.(V(5).OR.V(4)).AND.(V(2)
     *.NEQV.V(3)).AND.(V(3).EQV.V(4)).AND.(.NOT.(V(5).OR.
     *V(4).AND.V(1))))THEN
         :(Ausdrucken einer Lösung)
      END IF
10    CONTINUE
      ...
```

Ohne die Anweisungen Block IF und END IF läßt sich die Lösung natürlich auch mit der logischen IF-Anweisung formulieren, allerdings weniger bequem und übersichtlich:

```
      IF(.NOT.(e))GOTO 10
      (Ausdrucken einer Lösung)
      S=.TRUE.
10    CONTINUE
```

Wenn der Ausdruck e, der Meiers Aussagen enthält, den Wert "falsch" hat, wird also das Ausdrucken der Lösung durch GOTO 10 übersprungen.

Die Anwendung der logischen IF-Anweisung zur Lösung der Logelei wirkt bereits etwas schwerfällig. Der Eindruck verstärkt sich, wenn in Abhängigkeit von einem logischen Ausdruck entweder eine Anweisungsfolge A1 oder eine Anweisungsfolge A2 auszuführen ist. Im Prinzip kann man mit Hilfe der logischen IF- und der GOTO-Anweisung auch solche Programmstrukturen formulieren, die durch das Auftreten

mehrerer logischer Ausdrücke auf komplizierte Weise vernetzt sind, jedoch werden solche Programme durch die vielen Sprünge unübersichtlich. Die Norm hat in diesen Situationen durch Einführung der Anweisungen Block IF, END IF sowie ELSE IF und ELSE Abhilfe geschaffen.

Der IF-Level einer ausführbaren Anweisung st ist $n_1 - n_2$, wo n_1 die Anzahl der Block IF-Anweisungen vom Anfang der betreffenden Programmeinheit bis zur Anweisung st einschließlich, n_2 die Anzahl der END IF-Anweisungen vom Anfang der Programmeinheit bis zur Anweisung s ausschließlich ist. Er dient dem Compiler dazu, jeder Block IF-Anweisung ihre entsprechende END IF-Anweisung zuzuordnen: Diese ist die der Block IF-Anweisung folgende nächste END IF-Anweisung mit demselben IF-Level. Das zulässige Maximum des IF-Levels ist compilerabhängig und muß dem jeweiligen Handbuch entnommen werden. Die Norm macht hierzu keine Vorschrift.

Ein IF-Block besteht aus allen ausführbaren Anweisungen, die hinter einer Block IF-Anweisung stehen, bis zur nächsten ELSE IF-, ELSE- oder END IF-Anweisung ausschließlich, die denselben IF-Level wie die Block IF-Anweisung hat. Ein IF-Block kann leer sein.

Wenn der logische Ausdruck e einer Block IF-Anweisung den Wert "wahr" hat, werden die Anweisungen des IF-Blocks abgearbeitet. In einem IF-Block können weitere Block IF-END IF-Konstruktionen enthalten sein. Der IF-Block darf auch durch eine Verzweigungsanweisung verlassen werden. Wenn der IF-Block nicht durch eine Verzweigung verlassen wurde, wird das Programm hinter der END IF-Anweisung fortgesetzt, die denselben IF-Level wie die zum IF-Block gehörige IF-Anweisung besitzt. Dasselbe gilt, wenn e "wahr" und der IF-Block leer ist.

Wenn e den Wert "falsch" hat, wird der IF-Block nicht ausgeführt und das Programm hinter der nächsten END IF-, ELSE IF- oder ELSE-Anweisung fortgesetzt, die denselben IF-Level wie die Block IF-Anweisung hat.

Eine Block IF-Anweisung kann durch eine Marke gekennzeichnet sein, zu der verzweigt werden darf. Es ist aber verboten, unter Umgehung

der vorangehenden Block IF-Anweisung in einen IF-Block hineinzuspringen.

Die schon genannten Anweisungen ELSE IF und ELSE haben die Form

ELSE IF(e)THEN

wo e ein logischer Ausdruck ist, und

ELSE

Auf eine ELSE IF-Anweisung folgen die Anweisungen eines ELSE IF-Blocks. Dieser hat dieselbe Gestalt wie ein IF-Block und zu seiner Ausführung gilt das für IF-Blöcke Gesagte entsprechend.

Eine ELSE IF-Anweisung darf zwar eine Anweisungsnummer haben, aber es darf von keiner anderen Anweisung auf diese Bezug genommen werden. Es ist verboten, von außen unter Umgehung der zugehörigen ELSE IF-Anweisung in einen ELSE IF-Block zu verzweigen.

Auf eine ELSE-Anweisung folgen die Anweisungen eines ELSE-Blocks. Da dieser der letzte Block einer Block IF-END IF Konstruktion ist, muß auf ihn eine END IF-Anweisung folgen, die denselben IF-Level wie die zugehörige ELSE-Anweisung hat. Zu jeder Block IF-END IF-Konstruktion darf es höchstens eine ELSE-Anweisung geben. Hinsichtlich der Markierung der ELSE-Anweisung und der Verzweigungsverbote gilt das Entsprechende wie bei der ELSE IF-Anweisung und dem ELSE IF-Block.

Beispiel: Das Gleichungssystem

$$\begin{aligned} a\cdot x + b\cdot y &= c \\ d\cdot y &= e \end{aligned}$$

hat eine eindeutige Lösung, wenn $a\cdot d \neq 0$ ist, sonst gibt es entweder keine Lösung oder unendlich viele Lösungen, wobei der Lösungsraum die Dimension 1 oder 2 im mathematischen Sinn haben kann. Mögliche Kriterien für die Fälle, in denen es keine eindeutige Lösung gibt, sind:

1. Wenn mindestens eine der beiden Gleichungen widersprüchlich ist oder die Gleichungen miteinander unverträglich sind, gibt es keine Lösung. Dieser Fall liegt vor, wenn gilt: $a\cdot d = 0$ sowie $c \neq 0$

und a = b = 0 oder e ≠ 0 und d= 0 oder c·d - e·b ≠ 0.
2. Wenn alle Koeffizienten verschwinden, ist jedes Paar reeller Zahlen Lösung. Die Dimension des Lösungsraumes ist 2.
3. Wenn weder Fall 1 noch Fall 2 vorliegt, hat der Lösungsraum die Dimension 1.

Eine Block IF-END IF-Konstruktion kann daher lauten

```
      IF(A*D.NE.0)THEN
         Y=E/D
         X=(C-B*Y)/A
         PRINT*,'EINDEUTIGE LOESUNG, X=',X,' Y=',Y
      ELSE IF(C.NE.0..AND.A.EQ.0..AND.B.EQ.0..OR.
     *   E.NE.0..AND.D.EQ.0..OR.B*E-D*C.NE.0.)THEN
         PRINT*,'ES GIBT KEINE LOESUNG'
      ELSE IF(A.EQ.0..AND.B.EQ.0..AND.C.EQ.0..AND.
     *   D.EQ.0..AND.E.EQ.0.)THEN
         PRINT*,'JEDES PAAR REELLER ZAHLEN IST LOESUNG'
      ELSE
         PRINT*,'LOESUNGSRAUM HAT DIMENSION 1'
      END IF
      ...
```

Durch ein schematisches Beispiel soll noch darauf hingewiesen werden, daß ein Problem unterschiedlich gelöst werden kann, wobei auch die maximalen IF-Level verschieden sein können.

Angenommen, in Abhängigkeit von der Anordnung dreier paarweise ungleicher Elemente a, b und c werde je eine von 6 Anweisungsfolgen A1, A2,..., A6 ausgewählt gemäß der Zuordnungsvorschrift

Anordnung	Anweisungsfolge
a < b < c	A1
a < c < b	A2
b < a < c	A3
b < c < a	A4
c < a < b	A5
c < b < a	A6

Ein dieser Zuordnung entsprechender schematischer Ausschnitt eines

Programms kann lauten

```
                                                        IF-Level
...                                                        0
IF(a.LT.b.AND.b.LT.c)THEN                                  1
   ...A1...                                                1
ELSE IF(a.LT.c.AND.c.LT.b)THEN                             1
   ...A2...                                                1
ELSE IF(b.LT.a.AND.a.LT.c)THEN                             1
   ...A3...                                                1
ELSE IF(b.LT.c.AND.c.LT.a)THEN                             1
   ...A4...                                                1
ELSE IF(c.LT.a.AND.a.LT.b)THEN                             1
   ...A5...                                                1
ELSE                                                       1
   ...A6...                                                1
END IF                                                     1
...                                                        0
```

In der nächsten Version steigt der IF-Level um eine Stufe an.

```
                                                        IF-Level
...                                                        0
IF(a.LT.b.AND.a.LT.c)THEN                                  1
   IF(b.LT.c)THEN                                          2
      ...A1...                                             2
   ELSE                                                    2
      ...A2...                                             2
   END IF                                                  2
ELSE IF(b.LT.a.AND.b.LT.c)THEN                             1
   IF(a.LT.c)THEN                                          2
      ...A3...                                             2
   ELSE                                                    2
      ...A4...                                             2
   END IF                                                  2
ELSE                                                       1
   IF(a.LT.b)THEN                                          2
      ...A5...                                             2
```

```
   ELSE                                    2
      ...A6...                             2
   END IF                                  2
END IF                                     1
...                                        0
```

4 Unterprogramme

Es gibt folgende Arten von Unterprogrammen

(1) Innere Standardfunktionen
(2) Anweisungsfunktionen
(3) externe Funktionen
(4) Subroutinen
(5) BLOCK DATA-Unterprogramme

Außerdem wurde in Abschnitt 2.3 in einem Beispiel gezeigt, daß in derselben Programmeinheit ein Unterprogramm definiert und mit der zugeordneten GOTO-Anweisung aufgerufen werden kann.

Die Unterprogramme der Arten (1) bis (4) werden durch ihre Namen aufgerufen. Sie heißen Prozeduren. BLOCK DATA-Unterprogramme sind nicht aufrufbar, enthalten ausschließlich nicht ausführbare Anweisungen und werden in Abschnitt 8.5 behandelt.

Es gibt Unterprogramme, die der Programmierer außerhalb und solche, die er innerhalb der Programmeinheit definiert, in der sie aufgerufen werden. Man unterscheidet daher externe und interne Unterprogramme. Die Unterprogramme der Arten (3), (4) sind extern, die der Art (2) sind intern. Auch das Unterprogramm aus Abschnitt 2.3 ist intern.

Die drei Arten von Funktionen (1), (2), (3) haben das gemeinsame Merkmal, als Ergebnis einen Wert zu erzeugen. Subroutinen sind die vierte Art von Prozeduren, dazu bestimmt, allgemeinere Ergebnisse als nur einen Wert zu liefern.

Viele häufig benötigte Funktionen werden dem Progammierer als innere Standardfunktionen (intrinsic functions) zur Verfügung gestellt.

Eine Anweisungsfunktion wird durch eine einzige Anweisung erklärt. Sie kann nur in der Programmeinheit aufgerufen werden, in der sie definiert ist.

Externe Funktionen werden zusammen mit den Subroutinen externe Prozeduren genannt.

Eine oder mehrere externe Funktionen bilden ein Funktionsunterprogramm. Eine oder mehrere Subroutinen bilden ein Subroutine-Unterprogramm.

Meistens besteht ein Funktionsunterprogramm aus einer einzigen externen Funktion, ein Subroutine-Unterprogramm aus einer einzigen Subroutine. Ihre Ausführung beginnt dann an ihrem Anfang, d.h. mit der ersten ausführbaren Anweisung. Ein Funktionsunterprogramm kann mehrere externe Funktionen enthalten, ein Subroutine-Unterprogramm mehrere Subroutinen. Wenn das der Fall ist, was in der Regel nur bei umfangreichen Unterprogrammen vorkommt, so sind die Anfänge dieser Funktionen bzw. Subroutinen durch Eingangspunkte definiert (siehe 4.5). Jeder Eingangspunkt benennt eine weitere externe Funktion oder eine Subroutine.

Gewöhnlich werden Anwendungsprogramme aus verschiedenen Fachgebieten als externe Prozeduren in Programmbibliotheken zur Verfügung gestellt.

Ferner gibt es nicht in der Norm vorgesehene compilerabhängige Hilfsprozeduren (Service Subroutinen). Durch sie können bei Überläufen, Unterläufen und Divisionen durch Null geeignete Maßnahmen vorgesehen und ggf. Fehlerabbrüche vermieden werden.

Da Funktions- und Subroutine-Unterprogramme selbständige Programmeinheiten sind, ist es möglich, ein Programm aus unabhängigen Abschnitten zusammenzusetzen, die getrennt erstellt und ausgetestet werden können. Größere Programme sollten im wesentlichen aus solchen Unterprogrammen bestehen. Ein Hauptprogramm bildet dann nur einen Überbau, der erforderlich ist, um mehrere Programmeinheiten zu einem ausführbaren Programm zu vereinen.

4.1 Funktionen

Allen Funktionsarten gemeinsam ist ihr Aufruf

```
f([a[,a]...])
```

Hierin sind

f Name einer Funktion oder formaler Parameter, der den Namen einer Funktion vertritt

a aktuelle Parameter. Sie dienen der Übergabe von Daten

Der Aufruf einer Funktion ist ein Ausdruck. Welche Größen als aktuelle Parameter zugelassen sind, hängt von der Funktionsart ab.

4.1.1 Innere Standardfunktionen

Für den Programmierer wäre die Erstellung von Hilfsprogrammen zur Berechnung häufig benötigter Funktionen wie der Logarithmus- oder der Exponentialfunktion oder der trigonometrischen Funktionen sehr mühsam und schwierig. FORTRAN stellt daher für diese, aber auch für sehr simple Probleme innere Standardfunktionen zur Verfügung. Beispielsweise kann man zur Ermittlung des Restes der Division zweier INTEGER-Variablen I, J den Ausdruck I-I/J*J hinschreiben. Es gibt aber auch die Standardfunktion MOD, deren Aufruf MOD(I,J) dem oben genannten Ausdruck äquivalent ist.

Alle inneren Standardfunktionen sind in Anhang B angegeben. Als aktuelle Parameter sind Ausdrücke der angegebenen Typen zugelassen.

Ein Anwendungsbeispiel ergibt die Aufgabenstellung, näherungsweise die Nullstelle der Funktion $f(x) = x - \cos x$ zu berechnen. Zur Lösung können viele Verfahren vorgeschlagen werden. Beispielsweise kann der Fixpunkt der Abbildung $x \mapsto \cos x$ nach den folgenden Vorschriften gesucht werden:

1. Gib eine Anfangsnäherung x_0 und eine reelle Zahl eps vor.
2. Berechne eine verbesserte Näherung $x = \cos x_0$.
3. Wenn $|x - x_0| > eps$, setz $x = x_0$ und führ 2. aus, sonst stop.

Bild 3 veranschaulicht das Verfahren. Es läuft auf die näherungsweise Ermittlung des Schnittpunkts der Geraden $y = x$ und der Kurve $y = \cos x$ hinaus.

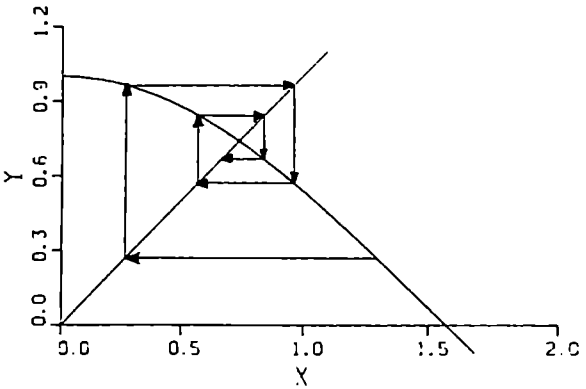

Bild 3

Dem Anhang B ist zu entnehmen, daß es die Funktion COS zur Berechnung eines Funktionswertes der Kosinusfunktion, die Funktion ABS zur Ermittlung des Absolutbetrags einer Zahl gibt. Ein Programm für die genannte Aufgabe kann lauten

```
      READ*,XO,EPS
12    X=COS(XO)
C     ZU JEDEM ITERATIONSSCHRITT WIRD DIE VERBESSERTE
C     NÄHERUNG AUSGEGEBEN
      PRINT*,X
      IF(ABS(X-XO)-EPS)11,10,10
10    XO=X
      GOTO 12
11    END
```

Für viele Standardfunktionen gibt es außer ihren speziellen Funktionsnamen auch Gattungsnamen, die auf Gruppen von Funktionen zugreifen. Dem Anhang B entnimmt man beispielsweise, daß es für die Kosinusfunktion die Standardfunktionen COS, DCOS, CCOS für aktuelle Parameter der Typen REAL, DOUBLE PRECISION, COMPLEX gibt. Sie ermitteln ein Ergebnis vom Typ des Parameters.

Es ist aber auch möglich, einen aktuellen Parameter der genannten

Typen mit dem Gattungsnamen COS zu verbinden. Der Typ des Parameters reicht dann dem Compiler als Information für die Auswahl des richtigen Unterprogramms aus.

In den meisten Fällen ist beim Aufruf einer Standardfunktion durch einen Gattungsnamen der Typ des Ergebnisses derselbe wie der Typ des aktuellen Parameters. Ausnahmen bilden u.a. die Standardfunktionen zur Typumwandlung sowie NINT (nearest INTEGER) und ABS mit komplexem aktuellen Parameter.

Wenn eine Standardfunktion mehr als einen Parameter hat, dann müssen diese beim Aufruf von demselben Typ sein.

Die Namen der Standardfunktionen sind lokal bezüglich der Programmeinheit, in der sie aufgerufen werden. Jeder dieser Namen kann daher statt zur Bezeichnung einer Standardfunktion dazu verwendet werden, eine beliebige andere lokale Größe zu benennen. In den anderen Programmeinheiten bleibt die ursprüngliche Bedeutung der Namen erhalten. Es ist aber auch möglich, die Namen der Standardfunktionen umzufunktionieren, indem sie zur Bezeichnung globaler Größen, z.B. mittels einer EXTERNAL-Anweisung, verwendet werden (siehe 4.3.3).

Eine IMPLICIT-Anweisung verändert den Typ einer Standardfunktion nicht. Die Angabe einer Typanweisung ist zulässig, aber überflüssig.

4.1.2 Anweisungsfunktionen

Diese Funktionen werden vom Programmierer durch eine einzige Anweisung der Form

```
f([d[,d]...])=e
```

definiert. Hierin sind

d Variablen als formale Parameter einer Anweisungsfunktion. Sie sind Stellvertreter von aktuellen Parametern, die beim Aufruf der Funktion benutzt werden

e Ausdruck

Die Definition ähnelt einer Wertzuweisung. Sie vereinbart aber eine Funktion und ist nicht ausführbar. Sie muß hinter den Spezifikationsanweisungen und vor der ersten ausführbaren Anweisung einer Programmeinheit stehen. Der Compiler kann die Definition einer Anweisungsfunktion nicht mit einer Wertzuweisung verwechseln, in der links vom Gleichheitszeichen ein Feldelement steht. Wenn diese Bedeutung vorläge, müßte eine Vereinbarung eines entsprechenden Feldes vorhanden sein. Eine Anweisungsfunktion kann nur in der Programmeinheit aufgerufen werden, in der sie definiert ist. Sie ist daher lokal bezüglich ihrer Programmeinheit.

Die formalen Parameter müssen in dem Ausdruck e erscheinen. Er kann weitere Operanden enthalten, deren Werte beim Aufruf der Anweisungsfunktion definiert sein müssen.

Durch Typ- oder IMPLICIT-Anweisung können für den Namen einer Standardfunktion und für ihre formalen Parameter Typen erklärt werden, sonst gilt die Namensregel. Selbstverständlich dürfen auch in dem Ausdruck e Operanden der Typen COMPLEX und DOUBLE PRECISION nicht verknüpft werden.

Für formale Parameter wird kein Speicherplatz reserviert. Die aktuellen Parameter müssen ihnen in Anzahl, Reihenfolge und Typ entsprechen.

Beim Aufruf einer Funktion sind als allgemeinste aktuelle Parameter Ausdrücke entsprechenden Typs zugelassen. Nach Auswertung werden sie anstelle der formalen Parameter in den Ausdruck e eingesetzt; danach wird dieser ausgewertet. Wenn das Ergebnis von einem anderen arithmetischen Typ ist als der symbolische Name der Funktion, wird es in den Typ der Funktion umgewandelt. Beispiele:

```
(1)     F(X,Y)=X+Y    (Definition der Anweisungsfunktion F)
        ...
        A=F(U*V,V)
        ...
```

Die letztgenannte Anweisung ist gleichbedeutend mit

```
        A=U*V+V
```

Eine sinnvolle Anwendungsmöglichkeit für die Anweisungsfunktion liegt offensichtlich vor, wenn Ausdrücke gleicher Bauart mit verschiedenen Operanden mehrmals in einer Programmeinheit auszuwerten sind.

(2) Die Implikation wird durch eine Anweisungsfunktion "WENN" implementiert und es wird eine Tabelle ausgegeben. Man beachte, daß der Name der Funktion sowie formale und aktuelle Parameter in einer Typanweisung erscheinen.

```
      LOGICAL WENN,A,B,X,Y
      WENN(A,B)=.NOT.A.OR.B
      DO 10 I=0,1
      X=I.EQ.1
      DO 10 J=0,1
      Y=J.EQ.1
10    PRINT*,'X=',X,' Y=',Y,' WENN(X,Y)=',WENN(X,Y)
      END
```

Die formalen Parameter werden nur zur Definition der Anweisungsfunktion benötigt. Die dafür verwendeten Namen können in anderen Anweisungen auch Variablen gleichen Typs bezeichnen. Die Namen formaler Parameter einer Anweisungsfunktion heißen daher lokal bezüglich der Definition der Anweisungsfunktion. In dem Beispiel können daher die Namen A, B auch als aktuelle Parameter anstelle X, Y benutzt werden.

(3)
```
      DIMENSION A(3,7)
      A1(J)=A(1,J)
      A2(J)=A(2,J)
      A3(J)=A(3,J)
      ...
```

Hier werden drei Anweisungsfunktionen A1, A2, A3 definiert, die auf die Zeilen des Feldes A zugreifen.

(4) Der Norm gemäß darf eine Anweisungsfunktion als Ausdruck aufgerufen werden. Sofern der Ausdruck e in einer Anweisungsfunktion nur aus einer Variablen, einem Feldelement oder einer Teilkette (siehe 5.1.3) besteht, lassen manche Compiler den Aufruf der Funktion auch

auf der linken Seite einer Wertzuweisung zu. Da in diesem Fall die Anweisungsfunktion anstelle einer Variablen, eines Feldelements oder einer Teilkette erscheint, ist es konsequent, sie auch entsprechend zu behandeln.

Eine geeignete Aufgabenstellung für ein Beispiel ist, die Elemente einer oberen n*n-Dreiecksmatrix (a_{ij}), $i \leq j$; i,j=1,...,n platzsparend in einem eindimensionalen Feld zu speichern, sie aber über eine Anweisungsfunktion in einer einem zweidimensionalen Feld entsprechenden Art aufzurufen.

Es muß dazu eine Funktion vorgegeben werden, welche die Werte der Indexpaare (i,j) der n*n-Dreiecksmatrix auf die Werte des Index eines eindimensionalen Feldes abbildet. Beispielsweise leistet dies die Funktion s(i,j) = (2*n - i)*(i - 1)/2 + j. Also kann das Programm für 10*10-Matrizen lauten

```
      PARAMETER (N=10,M=N*(N+1)/2)
      DIMENSION X(M)
      DRK(I,J)=X((2*N-I)*(I-1)/2+J)
      ...
C     IN BELIEBIGER WEISE KÖNNEN DEN MATRIXELEMENTEN
C     UEBER DIE FUNKTION DRK WERTE ZUGEWIESEN WERDEN,
C     Z.B. DEM ELEMENT IN DER 2. ZEILE UND 8. SPALTE
C     DER WERT 10 DURCH
      DRK(2,8)=10.
      ...
```

4.1.3 Externe Funktionen

Ein Funktionsunterprogramm ist eine selbständige Programmeinheit und definiert mindestens eine externe Funktion. Durch ENTRY-Anweisungen (siehe 4.5) können in einem Funktionsunterprogramm mehrere externe Funktionen definiert werden. Die erste Anweisung eines Funktionsunterprogramms lautet

```
[t] FUNCTION f([d[,d]...])
```

Hierin sind

t Typ der Funktion. Außer den bisher abgehandelten Typen kann auch CHARACTER[*1] erscheinen (siehe 5.1.1)

f Name des Funktionsunterprogramms. Er ist der Name einer externen Funktion

d Namen formaler Parameter

Die formalen Parameter sind wie bei einer Anweisungsfunktion Stellvertreter für aktuelle Parameter. In einer Funktion können aber außer Variablen auch Felder oder Prozeduren formale Parameter sein.

Man beachte, daß selbst dann, wenn keine Parameter übergeben werden, in der FUNCTION-Anweisung die Klammern angegeben werden müssen.

Auf die FUNCTION-Anweisung folgen zunächst, falls erforderlich, Spezifikationsanweisungen. Wenn ein formaler Parameter ein Feld bezeichnet, so muß es durch eine Felddefinition dimensioniert werden. Die Indexgrenzen müssen darin nicht notwendig konstante Ausdrücke sein, wie es bisher gefordert wurde. Sonst können alle Spezifikationsanweisungen erscheinen, insbesondere Typanweisungen für formale Parameter. Auch für den Namen der Funktion kann kann der Typ in einer Typ- oder IMPLICIT-Anweisung bestimmt werden.

Der Name des Unterprogramms wird im Unterprogramm wie eine Variable gebraucht. Er oder ein Name des gleichen Typs, der auf denselben Speicherplatz (oder auf dieselben Speicherplätze) zugreift (siehe 4.5, 5.2) muß mindestens einmal auf der linken Seite einer Wertzuweisung erscheinen. In den meisten Fällen sind gewissermaßen als Rumpf eines Funktionsunterprogramms neben der Wertzuweisung an den Funktionsnamen und der Anweisung END weitere ausführbare Anweisungen vorhanden. Die letzte Anweisung jedes Funktionsunterprogramms ist END. Sie informiert den Compiler über das Ende einer Programmeinheit und bewirkt bei der Ausführung den Rücksprung in die rufende Programmeinheit.

Die Namen externer Funktionen sind global; sie dürfen weder mit anderen globalen Namen (siehe 2.2.1) identisch sein und mit Ausnahme einer gleichnamigen Variablen auch nicht mit lokalen Namen der betreffenden Programmeinheit.

Die aktuellen Parameter müssen den formalen in Reihenfolge, Anzahl und Typ entsprechen. Ausnahme: Subroutinen (siehe 4.2) haben keine Typen.

Einer Variablen als formalem Parameter entspricht eine Variable oder ein Ausdruck als aktueller Parameter (siehe 4.3.1). Für Textausdrücke gibt es eine Einschränkung (siehe 5.2.6).

Einem Feld als formalem Parameter entspricht ein Feld oder ein Feldelement als aktueller Parameter. Sich daraus ergebende Fragen werden in Abschnitt 4.3.2 behandelt.

Einer Prozedur als formalem Parameter entspricht als aktueller Parameter eine Standardfunktion, eine externe Prozedur oder eine formale Prozedur. Das wird in Abschnitt 4.3.3 näher erklärt. Eine Anweisungsfunktion kann kein aktueller Parameter sein.

Beispiele:

(1) Es wird ein Funktionsunterprogramm zu der Aufgabe angegeben, näherungsweise $a^{1/3}$, $a \in R$, nach dem Newton-Verfahren zu berechnen, wobei eine geeignete Anfangsnäherung gewählt wird. Eingabeparameter sind a und eine Genauigkeitsschranke eps > 0. Das Verfahren läuft in folgenden Schritten ab:

1. Setz $x_0 = \begin{cases} \text{sign}(1,a), \text{ wenn } |a| > 1 \text{ (sign wie Standard-} \\ \qquad\qquad\qquad\qquad\qquad \text{funktion SIGN),} \\ a \text{ sonst.} \end{cases}$
2. Berechne $x = (2x_0 + a/x_0^2)/3$ (Verbesserung von x_0 nach Newton).
3. Wenn $|x - x_0| > eps$, dann setz $x = x_0$ und führ 1. aus, sonst Zuweisung von x an den Namen der Funktion und Rücksprung.

Ein Funktionsunterprogramm dazu kann lauten

```
      FUNCTION DW(A,EPS)
      IF(A.LE.1.)THEN
         XO=SIGN(1.,A)
      ELSE
         XO=A
      END IF
    1 X=(2.*XO+A/XO**2)/3
```

```
      IF(ABS(X-XO).LE.EPS)GOTO 2
      XO=X
      GOTO 1
2     DW=X
      END
```

In diesem Beispiel reserviert der Compiler Speicherplätze nur für das Ergebnis unter dem Namen DW sowie die Hilfsvariablen mit den Namen XO, X. Obwohl für formale Parameter kein Speicherplatz reserviert wird, müssen dem Compiler die Typen der formalen Parameter bekannt sein, natürlich nur insoweit, als es sich nicht um Namen von Subroutinen handelt. Es sind nämlich beispielsweise die arithmetischen Operationen durch Mikroprogramme hardwaremäßig realisiert, und von diesen müssen beim Übersetzen die richtigen ausgewählt werden. Es ist etwa klar, daß eine REAL-Addition ein anderes Mikroprogramm erfordert als eine INTEGER-Addition.

(2) Beim Aufruf einer externen Funktion muß der rufenden Programmeinheit deren Typ bekannt sein. Ein Programm zu der Aufgabe, den Mittelwert von 100 in einem Feld gespeicherten Meßwerten in einem Funktionsunterprogramm zu berechnen, kann lauten

```
      DIMENSION A(100)
C     DAS HAUPTPROGRAMM RUFT DAS FUNKTIONSUNTERPROGRAMM
C     MITTEL AUF, DAS DEN TYP REAL HAT. DAHER IST EINE TYP-
C     SPEZIFIKATION FUER MITTEL ERFORDERLICH
      REAL MITTEL
      .
      .(Eingabe der 100 Meßwerte)
      .
      PRINT*,'MITTELWERT=',MITTEL(A)
C     OHNE DIE TYP-ANWEISUNG FUER MITTEL WUERDE DIE IN MIT-
C     TEL ALS ERGEBNIS GESPEICHERTE BITKETTE ALS INTEGER-
C     ZAHL INTERPRETIERT UND ES WUERDE UNSINN AUSGEGEBEN
C     WERDEN
      END
      REAL FUNCTION MITTEL(A)
      DIMENSION A(100)
      MITTEL=0.
```

```
      DO 10 I=1,100
10    MITTEL=MITTEL+A(I)
      MITTEL=MITTEL/100.
      END
```

4.2 Subroutine-Unterprogramme

Diese Unterprogramme werden verwendet, wenn als Ergebnis kein Wert zu erzeugen ist, wie in einem Druck-Unterprogramm, oder mehr als nur ein Wert, wie beispielsweise in dem Fall, daß die Lösung eines linearen Gleichungssystems, also ein Vektor, zu berechnen ist. Im Prinzip kann das natürlich auch ein Funktionsunterprogramm leisten, in dem die Feldelemente des Lösungsvektors auf der linken Seite einer Wertzuweisung erscheinen und in dem das Feld in der Parameterliste enthalten ist, um das Ergebnis an die rufende Programmeinheit zu übergeben. Das ist jedoch kein sinnvoller Gebrauch eines Funktionsunterprogramms. Angemessen in solchen Fällen ist die Verwendung von Subroutine-Unterprogrammen. Sie werden manchmal auch eigentliche Unterprogramme genannt.

Die erste Anweisung eines Subroutine-Unterprogramms lautet

```
SUBROUTINE sn[([d[,d]...])]
```

Hierin sind

sn	Name des Subroutine-Unterprogramms
d	formale Parameter, die Namen von Variablen, Feldern oder Prozeduren sein können oder das Zeichen *(Stern)

Der Name sn ist global. In dem Unterprogramm darf er nur in der SUBROUTINE-Anweisung erscheinen, also auch nicht, wie in Funktions-Unterprogrammen, auf der linken Seite einer Wertzuweisung. Die Übergabe von Ergebnissen erfolgt daher entweder über Parameter oder über COMMON-Blöcke (siehe 8.1). Wenn von dieser alternativen Übergabemöglichkeit ausschließlich Gebrauch gemacht wird, entfällt die Parameterliste in der SUBROUTINE-Anweisung. Auch das Klammerpaar kann fehlen. Denkbar sind natürlich auch Unterprogramme, bei denen keine Parameter zu übergeben sind.

Ein Stern als formaler Parameter ermöglicht in Verbindung mit der RETURN-Anweisung (siehe 4.4) einen Rücksprung zu einer beliebigen ausführbaren Anweisung der rufenden Programmeinheit.

Spezifikationsanweisungen müssen den ausführbaren Anweisungen vorangehen. Die letzte Anweisung jedes Subroutine-Unterprogramms ist END.

Beispiel: Angenommen, die Bundesbank bringe 3-DM-Stücke, genannt Taler, in den Verkehr. Das Subroutine-Unterprogramm TALER ermittelt, wie ganzzahlige DM-Beträge von mehr als 7 DM mit 3-DM-Stücken und höchstens mit zwei 5-DM-Stücken gezahlt werden können. Bei der Division des DM-Betrages durch 3 kann ein von Null verschiedener Divisionsrest nur 1 oder 2 sein. Er kann für Beträge von mehr als 7 DM durch zwei 5-DM-Stücke bzw. ein 5-DM-Stück abgefangen werden.

```
      SUBROUTINE TALER(Z,A,B)
C     Z IST EINGABEPARAMETER FUER DEN BETRAG, A ANZAHL
C     DER 3-DM-STUECKE, B ANZAHL DER 5-DM-STUECKE
      INTEGER Z,A,B
      IF(Z.LE.7)THEN
      PRINT*,'BETRAG VON DM',Z,' ZU KLEIN'
      ELSE
      B=MOD(3-MOD(Z,3),3)
      A=(Z-5*B)/3
      END IF
      END
```

Der Aufruf einer Subroutine erfolgt durch die Anweisung

```
CALL sn[([a[,a]...])]
```

Hierin sind

sn	Name einer Subroutine
a	aktuelle Parameter

4.3 Formale und aktuelle Parameter

Bei der Übergabe von Daten zwischen rufender und gerufener Pro-

grammeinheit unterscheidet man Eingabe- und Ausgabeparameter. Ist ein formaler Parameter ein Feld (siehe 4.3.2) oder eine Prozedur (siehe 4.3.3), so ergeben sich Regeln aus der Tatsache, daß in ausführbaren Maschinenprogrammen aktuelle Parameter durch Adressen repräsentiert werden. Eine Adresse kann eine Variable, das erste Element eines Feldes, des Anfang eines Unterprogramms oder eine Rücksprungstelle (siehe 4.4) kennzeichnen.

4.3.1 Ein- und Ausgabeparameter, Nebeneffekte

Parameter, die dazu verwendet werden, einem Unterprogramm Daten zur Verarbeitung zu übergeben, werden als Eingabeparameter bezeichnet. Entsprechend werden über Ausgabeparameter im Unterprogramm ermittelte Ergebnisse an die rufende Programmeinheit übergeben.

Ein formaler Parameter, der als Ein- oder Ausgabeparameter verwendet wird, ist eine Variable oder ein Feld. Gewöhnlich ist ein solcher Parameter entweder Eingabe- oder Ausgabeparameter, aber nicht beides gleichzeitig.

Da Funktionen primär dazu bestimmt sind, nur einen Wert als Ergebnis zu ermitteln und an die rufende Programmeinheit zu übergeben, sind gewöhnlich die Elemente ihrer Parameterliste Eingabeparameter. Im Prinzip ist es auch bis auf Ausnahmen, die am Ende dieses Abschnitts angegeben werden, zulässig, Ausgabeparameter in Funktionen zu benutzen; weil dann Nebeneffekte möglich sind, wird oft empfohlen, von dieser Möglichkeit keinen Gebrauch zu machen.

Im folgenden Beispiel wird ein Nebeneffekt dadurch hervorgerufen, daß ein formaler Parameter, der eine Variable ist, auf der linken Seite einer Wertzuweisung erscheint und gleichzeitig als Eingabe- und als Ausgabeparameter verwendet wird.

```
PROGRAM HAUPT
U=2.
V=3.
W1=U/V
W2=F(U,V)
```

```
      ...
C     ES IST JETZT W1=.666..., W2=2.5
      U=2.
      V=3.
      W2=F(U,V)
      W1=U/V
C     ES IST JETZT W1=.4, W2=2.5, WEIL F VOR DER BERECHNUNG
C     VON U/V AUFGERUFEN WURDE
      END
      FUNCTION F(X,Y)
      Y=X+Y
      F=Y/X
      END
```

Daß Ergebnisse nur von der Reihenfolge von Anweisungen abhängen, wie in diesem Beispiel, ist im allgemeinen nicht beabsichtigt.

Durch den Aufruf einer Funktion darf nicht der Wert einer Größe in derselben Anweisung verändert werden. Unzulässig wäre daher mit der oben definierten Funktion F die Anweisung

```
W=U/V+F(U,V)
```

Sie bleibt auch dann unzulässig, wenn V durch die Anweisung erst definiert wird.

Wenn in einer Anweisung mehr als eine Funktion aufgerufen wird, so hängt es vom Compiler ab, in welcher Reihenfolge die Funktionswerte berechnet werden. Die Funktionen dürfen dann aber keine Nebeneffekte bewirken, welche die Funktionswerte von der Reihenfolge ihrer Errechnung abhängig machen.

Ein Index eines Feldelements kann durch einen Indexausdruck vom Typ INTEGER (siehe 2.4) ermittelt werden, in dem Funktionen aufgerufen werden. Die obige Regel verbietet insbesondere Nebeneffekte, die andere Indizes desselben Feldeelements verändern.

Enthält ein Ausdruck Funktionsaufrufe als Operanden, so werden diese u.U nicht alle ausgewertet, wenn der Wert des Ausdrucks von den Funktionswerten unabhängig ist. Bei logischen Ausdrücken ist das besonders evident. Wenn z.B. in dem Ausdruck

```
X.GT.Y.OR.L(U,V)
```

wo L eine logische Funktion bezeichnet, der Vergleichsausdruck X.GT.Y den Wert "wahr" ergibt, braucht L(U,V) nicht errechnet zu werden. Wenn die U oder V entsprechenden formalen Parameter auf den linken Seiten von Wertzuweisungen stehen, werden Nebeneffekte unkontrollierbar. Entsprechendes gilt für alle Ausdrücke. Hat beispielsweise in einem arithmetischen Ausdruck ein Faktor den Wert Null und sind Funktionen als Operanden vorhanden, so kann ihre Auswertung unterbleiben.

Wenn in einer logischen IF-Anweisung

```
IF(e)st
```

im Ausdruck e eine Funktion auftritt, so wird sie vor einer eventuell in der Anweisung st enthaltenen Funktion ausgewertet. Wenn e den Wert "falsch" ergibt, wird eine Funktion in st in keinem Fall aufgerufen.

Nicht compilerabhängig ist die Reihenfolge der Funktionsauswertung, wenn ein Funktionsaufruf als Operanden eines aktuellen Parameters einen Funktionsaufruf enthält, z.B.

```
Y=F(G(X)+Z)
```

Hier wird selbstverständlich G(X) zuerst errechnet.

Ist ein formaler Parameter eine Variable und wird er als Eingabeparameter verwendet, so darf der entsprechende aktuelle Parameter ein Ausdruck sein. Wird der Parameter dagegen als Ausgabeparameter verwendet, so ist als aktueller Parameter nur eine Variable, ein Feldelement oder eine Teilkette (siehe 5.1.3) zulässig.

Zur Vermeidung von Nebeneffekten beim Gebrauch von Subroutinen wird empfohlen, denselben Parameter nicht gleichzeitig als Eingabe- und als Ausgabeparameter zu verwenden.

4.3.2 Felder als Parameter

Der Name des als formaler Parameter gebrauchten Feldes muß im Unterprogramm in einer Spezifikationsanweisung mit Felddefinition er-

scheinen. Dadurch wird kein Speicherplatz reserviert, aber der Compiler erkennt daraus, daß ein Feld bezeichnet wird und bildet einen Ausdruck zur Berechnung der Speicherabbildungsfunktion. Für die Felddefinition gibt es drei Möglichkeiten:

(1) Die Indexgrenzen sind konstante INTEGER-Ausdrücke.
(2) Die Indexgrenzen sind INTEGER-Ausdrücke, in denen außer Konstanten auch Variablen als Operanden vorkommen.
(3) Die obere Grenze des letzten Index wird dem Unterprogramm nicht mitgeteilt und in der Felddefinition durch einen Stern ersetzt. Diese Möglichkeit der Felddefinition erklärt sich dadurch, daß die obere Grenze des letzten Index von der Speicherabbildungsfunktion nicht benötigt wird.

Indexgrenzen der Form (2) oder (3) sind nur in Funktions- und Subroutine-Unterprogrammen möglich, nicht in Hauptprogrammen.

Einem formalen Parameter, der symbolischer Name eines Feldes ist, entspricht als aktueller Parameter der Name eines Feldes oder eines Feldelements. Zur Abkürzung wird von den Begriffen formales und aktuelles Feld Gebrauch gemacht. Dadurch darf aber nicht die Tatsache verschleiert werden, daß nur ein aktuelles Feld physikalisch existiert.

Aus der Tatsache, daß die Elemente des aktuellen Feldes konsekutive Worte des Hauptspeichers belegen und aus einem Unterprogramm heraus auf eine zusammenhängende Teilmenge dieses Abschnitts zugegriffen wird, ergeben sich einige Folgerungen.

Größe und Anzahl der Dimensionen des aktuellen und des formalen Feldes können verschieden sein. Wenn der aktuelle Parameter ein Feld ist, so entspricht das erste Element des aktuellen Feldes dem ersten Element des formalen Feldes, das zweite Element des aktuellen Feldes dem zweiten Element des formalen Feldes usw..

Wenn der aktuelle Parameter ein Feldelement ist, so entspricht dieses Element dem ersten Element des formalen Feldes, das folgende nächste Element des aktuellen Feldes dem zweiten Element des formalen Feldes usw..

Es ist also kein Unterschied, ob als aktueller Parameter das aktu-

elle Feld oder sein erstes Element genannt wird.

Das formale Feld muß in dem aktuellen Feld enthalten sein, damit nicht auf andere, nicht zum aktuellen Feld gehörige Worte zugegriffen wird.

Einige Beispiele sollen die verschiedenen Möglichkeiten der Felddefinition in externen Prozeduren erläutern.

(1) In einem 10*10-Feld soll das Element mit dem größten und das mit dem kleinsten Wert gesucht werden. Da das Feld konsekutive Plätze im Hauptspeicher belegt, kann man mit einer DO-Anweisung bei der Suche auskommen, wenn man sie in einem Unterprogramm auf einem eindimensionalen formalen Feld formuliert. Das Programm kann dann die Gestalt haben:

```
      DIMENSION A(10,10)
      READ*,A
      CALL MINMAX(A)
      END
      SUBROUTINE MINMAX(B)
      DIMENSION B(100)
      AMINI=B(1)
      AMAXI=B(1)
      DO 10 I=2,100
      IF(AMINI-B(I))20,10,21
21    AMINI=B(I)
      GOTO 10
20    IF(AMAXI-B(I))22,10,10
22    AMAXI=B(I)
10    CONTINUE
      PRINT*,'MIN=',AMINI,' MAX=',AMAXI
      END
```

Die Parameter A, B erscheinen also in Felddefinitionen mit konstanten Indexgrenzen, aber die Anzahl der Indizes und ihre Grenzen sind für A bzw. für B verschieden.

(2) Der Programmausschnitt aus Abschnitt 2.5 zur Lösung von Gleichungssystemen Ax = b, wo A eine n*n-Dreiecksmatrix ist, soll zu

einem Subroutine-Unterprogramm ergänzt werden, dem auch n als Parameter übergeben wird. Das Unterprogramm soll von einem Hauptprogramm aufgerufen werden, in dem Felder für A, x, b vereinbart sind und in dem die Eingabe der Daten erfolgt. Ein ausführbares Programm, das Systeme mit höchstens np=20 Gleichungen und Unbekannten löst, kann lauten

```
      PARAMETER (NP=20)
      DIMENSION A(NP,NP),B(NP),X(NP)
C     DIE ANZAHL DER GLEICHUNGEN UND UNBEKANNTEN SEI N
      READ*,N
      (Eingabe von A und b)
      CALL GGLS(A,B,X,N,NP)
      (Ausgabe der Lösung x)
      END
      SUBROUTINE GGLS(A,B,X,N,NP)
C     IN DER FELDDEFINITION FUER A WERDEN INDEXGRENZEN NP
C     WIE IN DER RUFENDEN PROGRAMMEINHEIT ANGEGEBEN, DAMIT
C     DIE SPEICHERABBILDUNGSFUNKTION RICHTIG GEBILDET WIRD
      DIMENSION A(NP,NP),B(NP),X(NP)
      X(N)=B(N)/A(N,N)
      :
10    X(I)=B(I)/A(I,I)
      END
```

Würden im Unterprogramm GGLS die Felder A, B, X mit

```
      DIMENSION A(N,N),B(N),X(N)
```

dimensioniert, so würde falsch auf die Speicherplätze von A zugegriffen werden, wenn nicht n=np=20 ist; die Zugriffe auf B und X würden sich nicht ändern, weil die obere Grenze des letzten Index von der Speicherabbildungsfunktion nicht benötigt wird.

(3) Ein Beispiel für den Fall, daß ein Feld formaler, ein Feldelement aktueller Parameter ist, ergibt sich aus der Aufgabenstellung, in einer Funktion für n*n-Matrizen die Euklidische Norm des k-ten Spaltenvektors zu berechnen. In einem Hauptprogramm soll für die Matrizen ein np*np-Feld vereinbart werden. Es sei $n \leq np$.

```
      PARAMETER(NP=...)
      DIMENSION A(NP,NP)
      :
      :(Eingabe von n, A)
      :
      DO 10 K=1,N
10    PRINT*,ENORM(A(1,K),N)
      END
      FUNCTION ENORM(SP,N)
      DIMENSION SP(N)
      ENORM=0.
      DO 10 J=1,N
10    ENORM=ENORM+SP(J)**2
      ENORM=SQRT(ENORM)
      END
```

(4) Man kann in den oben angegebenen Unterprogrammen auch durch

```
      DIMENSION A(NP,*),B(*),X(*)
```

bzw.

```
      DIMENSION SP(*)
```

dimensionieren, jedoch ist in den vorstehenden Beispielen dadurch nichts gewonnen. Im ersten Fall muß der Parameter NP übergeben werden, weil er zur Dimensionierung von A weiterhin erforderlich ist; im zweiten Fall wird der Parameter N in der DO-Anweisung benötigt. Unzulässig ist es, ein Feld, das in dieser Art erklärt wurde, in einer Ein-/Ausgabeliste zu nennen.

4.3.3 Prozeduren als Parameter

Eine externe Prozedur kann den Aufruf eines anderen Unterprogramms durch einen formalen Parameter als formale Prozedur vorsehen, die durch einen formalen Parameter benannt wird. Der entsprechende aktuelle Parameter ist der Name einer externen Prozedur, einer inneren Standardfunktion oder auch einer formalen Prozedur. Der zuletzt genannte Fall ergibt sich, wenn in einer Prozedur eine andere Prozedur aufgerufen wird, wobei eine formale Prozedur aktueller Para-

meter ist. Diese Situation findet man im nächsten Beispiel.

Damit der Compiler in der rufenden Programmeinheit Prozeduren als aktuelle Parameter von Variablen unterscheiden kann, müssen externe Prozeduren in der Anweisung

EXTERNAL p,[p]...

genannt werden. Hierin sind

p Namen von externen oder formalen Prozeduren. Ferner Namen von BLOCK-DATA-Unterprogrammen (siehe 8.5)

Die EXTERNAL-Anweisung kann auch erforderlich sein, um zwischen einer Standardfunktion und einer externen Funktion zu unterscheiden. Wenn eine Funktion aufgerufen wird, die keine Anweisungsfunktion ist und nicht in einer EXTERNAL-Anweisung genannt wird, so hat der Compiler zwischen zwei Möglichkeiten zu wählen: Es muß sich um eine Standardfunktion oder um eine externe Funktion handeln. Er prüft zuerst, ob die Funktion als Standardfunktion bekannt ist. Wenn das der Fall ist, wird diese benutzt; sonst unterstellt der Compiler, daß eine benutzerdefinierte externe Funktion gemeint ist. Ist aber die Funktion in einer EXTERNAL-Anweisung genannt, so wird sie ohne weitere Prüfung als externe Funktion aufgefaßt. Wenn eine externe Funktion benutzt werden soll, deren Name mit dem einer Standardfunktion identisch ist, so muß daher die externe Funktion in einer EXTERNAL-Anweisung genannt werden.

Da manche Compiler zusätzlich nicht normgemäße Standardfunktionen kennen, wird mitunter empfohlen, zur Vermeidung eines Fehlers alle externen Funktionen in EXTERNAL-Anweisungen zu nennen. Wenn ein Programm auf einer Anlage fehlerfrei gelaufen ist und es soll auf einer anderen Anlage mit zusätzlichen Standardfunktionen eingesetzt werden, dann wird durch diese Maßnahme vermieden, daß unbeabsichtigt anstelle einer externen Funktion eine Standardfunktion benutzt werden kann.

Die EXTERNAL-Anweisung soll nun an der Aufgabe erläutert werden, das Integral einer geeigneten Funktion f, $\int_a^b f(x)dx$, näherungsweise nach der Trapez- und nach der Simpson-Formel zu berechnen. Als Beispiel wird $f(x) = 3x^2 + 2x + 1$ genommen. Der Wert des Integrals ist

bekannt und kann mit dem numerischen Ergebnis verglichen werden, das durch das Programm ermittelt wird.

```
      EXTERNAL F
      READ*,A,B
C     DURCH DIE EXTERNAL-ANWEISUNG IST DEM COMPILER BE-
C     KANNT, DASS F EINE EXTERNE PROZEDUR UND WEDER EINE
C     VARIABLE NOCH EINE STANDARDFUNKTION IST. TRAPEZ UND
C     SIMPSO HABEN DEN TYP REAL GEMAESS NAMENSREGEL
      PRINT*,'INTEGRALE. TRAPEZ=',TRAPEZ(A,B,N,F),
     *' SIMPSO=',SIMPSO(A,B,N,F)
      END
      FUNCTION SIMPSO(A,B,N,G)
C     DA DIE FORMALE PROZEDUR G IM AUFRUF TRAPEZ ALS AKTU-
C     ELLER PARAMETER VERWENDET WIRD, MUSS G IN EINER
C     EXTERNAL-ANWEISUNG GENANNT WERDEN
      EXTERNAL G
      H=(B-A)/N
      SIMPSO=0.
      DO 10 I=1,N-1,2
10    SIMPSO=SIMPSO+G(A+I*H)
C     G HAT DEN TYP REAL GEMAESS NAMENSREGEL
      SIMPSO=2.*(H*SIMPSO+TRAPEZ(A,B,N,G))/3.
      END
      FUNCTION TRAPEZ(A,B,N,G)
C     DA DIE FORMALE PROZEDUR G NUR DIREKT AUFGERUFEN WIRD
C     UND NICHT ALS AKTUELLER PARAMETER VERWENDET WIRD, IST
C     DIE EXTERNAL-ANWEISUNG NICHT ERFORDERLICH
      H=(B-A)/N
      TRAPEZ=(G(A)+G(B))/2.
      DO 10 I=1,N-1
10    TRAPEZ=TRAPEZ+G(A+I*H)
      TRAPEZ=TRAPEZ*H
      END
      FUNCTION F(X)
      F=(3.*X+2.)*X+1.
      END
```

Wenn f(x) durch sin(x) zu ersetzen ist und zu berechnen durch die Standardfunktion mit dem speziellen Namen SIN, dann sind folgende Änderungen erforderlich: Das Funktions-Unterprogramm F ist natürlich überflüssig und im Hauptprogramm ist in den Aufrufen TRAPEZ und SIMPSO der aktuelle Parameter F durch SIN zu ersetzen. Die Anweisung EXTERNAL F muß durch INTRINSIC SIN ersetzt werden: Wird der Name einer inneren Standardfunktion als aktueller Parameter verwendet, so ist die Anweisung

```
INTRINSIC f[,f]...
```

erforderlich. Hierin sind

f Spezielle Namen von inneren Standardfunktionen zur Identifikation von aktuellen Parametern als Standardfunktionen. Ausgenommen sind aber die Typumwandlungsfunktionen INT, IFIX, IDINT, FLOAT, SNGL, REAL, DBLE, CMPLX, ICHAR, CHAR, die lexikographischen Funktionen LGE, LGT, LLE, LLT und die Funktionen zur Bestimmung eines Maximums oder Minimums MAX, MAXO, AMAX1, DMAX1, AMAXO, MAX1, MIN, MINO, AMIN1, DMIN1, AMINO, MIN1, weil sie nicht als aktuelle Parameter verwendet werden dürfen. Bei manchen Compilern kommen zu diesen Standardfunktionen weitere hinzu

Durch INTRINSIC SIN wird dem Compiler mitgeteilt, daß SIN eine innere Standardfunktion, also weder eine Variable noch eine externe Funktion ist. Durch den aktuellen Parameter SIN wird dann dem Unterprogramm die betreffende Anfangsadresse übergeben.

Die Anweisung EXTERNAL G ist beizubehalten. Dem Compiler muß nur bekannt sein, daß der aktuelle Parameter G im Aufruf TRAPEZ der Funktion SIMPSO eine Funktion und keine Variable bezeichnet. Ob es eine externe Funktion oder eine innere Standardfunktion ist, ist bedeutungslos, da die Anfangsadresse des Unterprogramms SIN beim Aufruf von SIMPSO im Hauptprogramm "durchgereicht" wird.

Nun geht auch aus Anweisung 10 der Funktion SIMPSO hervor, daß G eine Funktion und keine Variable ist. Daher ist bei manchen Compilern EXTERNAL G verzichtbar. Die Norm verlangt nicht, daß der Compiler wegen der Anweisung 10 G als Funktion erkennt.

Unzulässig ist der Aufruf einer Prozedur durch sich selbst, sowohl direkt, d.h der Aufruf erscheint in einer ausführbaren Anweisung des betreffenden Funktions- oder Subroutine-Unterprogramms, noch indirekt, indem etwa beim Aufruf ein formaler Parameter durch den Namen der Prozedur ersetzt wird oder indem ein Unterprogramm ein anderes Unterprogramm aufruft und dieses wiederum das erste Unterprogramm. Ein Unterprogramm, das sich selbst aufruft, wird gewöhnlich rekursiv genannt.

4.4 Außergewöhnlicher Rücksprung

Nach der Norm darf nur in Subroutinen ein Stern als formaler Parameter erscheinen, jedoch nicht in einer Funktion. Durch den entsprechenden aktuellen Parameter der Form *s wird eine Marke s übergeben, die in der rufenden Programmeinheit vorhanden sein muß. Die formalen Parameter *(Stern) in der Parameterliste werden von links mit 1 beginnend gezählt. In der Anweisung

```
RETURN [i]
```

ist

i ein INTEGER-Ausdruck

Zeigt i auf das i-te Zeichen *(Stern) in der Liste der formalen Parameter und ist der entsprechende aktuelle Parameter *s, so wird zu der durch s markierten Anweisung der rufenden Programmeinheit verzweigt. Fehlt i in der RETURN-Anweisung oder ist kein i-tes Zeichen in der Parameterliste vorhanden, so wird zu der nächsten ausführbaren Anweisung in der rufenden Programmeinheit verzweigt, die auf den Aufruf CALL sn[(...)] der Subroutine folgt, also ein "normaler" Rücksprung ausgeführt.

Um ein Beispiel angeben zu können, wird das Subroutine-Unterprogramm TALER aus Abschnitt 4.2 leicht verändert. Wenn eine Zahl kleiner oder gleich 7 übergegeben wurde, soll in der rufenden Programmeinheit zu der Anweisung mit der Nummer 100 verzweigt und BETRAG VON DM (Wert von Z) ZU KLEIN ausgegeben werden. Das Unterprogramm kann dann lauten

```
      SUBROUTINE TALER(Z,A,B,*)
      INTEGER Z,A,B
      IF(Z.LE.7)RETURN 1
      A=...
      B=...
      END
```

In der rufenden Programmeinheit können die Anweisungen stehen

```
      ...
      SUBROUTINE TALER(Z,A,B,*100)
      :
      :
100   PRINT*,'BETRAG VON DM',Z,' ZU KLEIN'
      ...
```

Eine RETURN-Anweisung ohne Ausdruck i ist auch in einer Funktion zugelassen.

4.5 ENTRY

Gewöhnlich beginnt die Abarbeitung eines Funktions- oder Subroutine-Unterprogramms bei der ersten ausführbaren Anweisung, die hinter der FUNCTION- bzw. SUBROUTINE-Anweisung erscheint. Man kann aber auch andere Eingangspunkte bestimmen durch die nicht ausführbare Anweisung

```
      ENTRY en[([d[,d]...])]
```

Hierin sind

en	Name eines Eingangspunktes
d	Namen von formalen Parametern

Die ENTRY-Anweisung kann sowohl in Funktions- als auch in Subroutine-Unterprogrammen verwendet werden mit der plausiblen Einschränkung, daß sie nicht in einem IF-Block erscheinen darf und nicht im Wiederholungsbereich einer DO-Anweisung. Die Liste der formalen Parameter einer ENTRY-Anweisung braucht in keiner Weise mit der Parameterliste der vorangehenden FUNCTION-oder SUBROUTINE-Anweisung noch der einer anderen ENTRY-Anweisung übereinzustimmen.

Durch jede ENTRY-Anweisung wird eine weitere externe Prozedur begonnen; innerhalb eines Funktionsunterprogramms ist es eine externe Funktion, innerhalb eines Subroutine-Unterprogramms ist es eine Subroutine. Die Namen der Eingangspunkte werden genauso wie die von Unterprogrammen verwendet, also im Falle externer Funktionen als Variablen, die von verschiedenen numerischen Typen sein dürfen. In Subroutine-Unterprogrammen erscheinen die Namen der Eingangspunkte nur in ENTRY-Anweisungen.

Die Variablen, auf die unter den Namen eines Funktionsunterprogramms oder seiner Eingangspunkte zugegriffen wird, belegen gleiche Speicherplätze, als ob sie in einer EQUIVALENCE-Anweisung (siehe S.2) gleichgesetzt wären. Wenn daher dem Namen eines Funktionsunterprogramms oder dem eines seiner Eingangspunkte gleichen Typs ein Wert zugewiesen wird, so werden alle diese Namen mit dem gleichen Wert definiert.

Um die Anwendung der ENTRY-Anweisung an einem Beispiel zu zeigen, kann man die oben angegebenen Unterprogramme SIMPSO und TRAPEZ zu einem einzigen Unterprogramm zusammenfassen. Da das Ergebnis der Trapezregel in jedem Falle zu berechnen ist, empfiehlt es sich, TRAPEZ als Eingangspunkt zu erklären und Anweisungen zu umgehen, die zur Berechnung der Simpson-Summe ausgeführt werden müssen. Die Zusammenfassung der Funktionsunterprogramme SIMPSO und TRAPEZ zu einem neuen Funktionsunterprogramm SIMPSO kann dann lauten

```
      FUNCTION SIMPSO(A,B,N,G)
      LOGICAL RETOUR
      RETOUR=.FALSE.
      GOTO 10
      ENTRY TRAPEZ(A,B,N,G)
      RETOUR=.TRUE.
10    H=(B-A)/N
      SIMPSO=(G(A)+G(B))/2.
      DO 11 I=1,N-1
11    SIMPSO=SIMPSO+G(A+I*H)
      SIMPSO=H*SIMPSO
      IF(RETOUR)RETURN
```

```
      DO 12 I=1,N-1,2
12    SIMPSO=SIMPSO+H*G(A+I*H)
      SIMPSO=2.*SIMPSO/3.
      END
```

TRAPEZ und SIMPSO benennen in diesem Programm dasselbe Wort und besitzen denselben Typ. Es ist daher gleichgültig, welchem der beiden Namen vor Verlassen des Unterprogramms das Ergebnis zugewiesen wird.

Da TRAPEZ in der neuen Programmversion Eingangspunkt ist, wird dieser Name wie der einer Variablen behandelt. Den Aufruf TRAPEZ(A,B,N,G) aus dem alten Programm SIMPSO darf man nicht übernehmen, da er hier rekursiv und damit verboten wäre.

Daß die Anweisung ENTRY in diesem Beispiel gegenüber der alten Version keinen großen Vorteil bringt, darf nicht verwundern. Sinnvolle Anwendungen von ENTRY erfolgen meistens in umfangreichen Unterprogrammen.

Wiederholt sind in den Beispielen dieses Abschnitts Hilfsgrößen benutzt worden, die in keiner Parameterliste erschienen. Während der Ausführung eines Unterprogramms werden ihnen Werte zugewiesen. Es hängt vom Rechner ab, ob nach dem Verlassen des betreffenden Unterprogramms die Plätze der Hilfsgrößen im allgemeinen mit anderen Werten überspeichert werden, so daß ihre Werte beim nächsten Aufruf des Unterprogramms undefiniert sind. Wenn überspeichert wird, kann man die Hilfsgrößen durch die Anweisung SAVE (siehe 8.5) auf zusätzlich bereitgestellte Speicherplätze retten.

5 Textverarbeitung

Die Programmiersprache FORTRAN wurde zunächst zur Lösung rein numerischer Probleme entwickelt. Deshalb sahen frühe Spachversionen keine oder nur sehr umständlich anzuwendende Möglichkeiten zur Textverarbeitung, d.h. zur Verarbeitung von Daten in einer externen Darstellung, vor.

Mit der neuen Norm wurde das grundlegend geändert. Wie zur Aufnahme von arithmetischen und logischen Werten können auch zur Aufnahme von Texten (strings), d.h. von Zeichenfolgen, Variable und Felder vereinbart werden, die kurz als Textvariable und Textfelder bezeichnet werden sollen. In einem Punkt besteht allerdings ein wesentlicher Unterschied: Bei arithmetischen und logischen Variablen und Feldelementen wird bereits durch den jeweiligen Typ eindeutig festgelegt, wie viele Speicherworte für die Variable bzw. das Feldelement benötigt werden. Bei Textvariablen und Elementen von Textfeldern kann dagegen der Programmierer wählen, für wie viele Zeichen sie Platz bieten sollen.

5.1 Textvariable und Textkonstanten

5.1.1 Das Schlüsselwort CHARACTER

Das Schlüsselwort, mit dem (benannte) Textgrößen vereinbart werden müssen, ist

```
CHARACTER [*l]
```

Hierin ist l die Länge der zu vereinbarenden Größe(n), d.h. die Anzahl der Zeichen, die die Größe(n) aufnehmen können soll(en). Man beachte: Bei Textfeldern erhält jedes Element die Länge l; alle Elemente eines Textfeldes besitzen immer die gleiche Länge.

Die Länge l kann sein

(1) eine positive INTEGER-Konstante ohne Vorzeichen,
(2) ein konstanter INTEGER-Ausdruck (siehe 2.1.6) mit positivem Wert, eingeschlossen in Klammern, oder
(3) ein Stern, eingeschlossen in Klammern (siehe 5.2.4).

Falls die Angabe *l fehlt, setzt der Rechner die Länge 1 ein.

Man beachte: Die Mindestlänge von Textgrößen ist 1. Eine maximale Länge wird durch die Norm zwar nicht vorgeschrieben, doch existieren in der Praxis Beschränkungen. So läßt VS FORTRAN von IBM 500 und FORTRAN (ASCII) von UNIVAC 511 als maximale Länge zu; VAX-11 FORTRAN von DEC läßt dagegen die maximale Länge 32767 zu.

Das Schlüsselwort CHARACTER[*l] kann wie die anderen Schlüsselwörter, die Datentypen bezeichnen, verwendet werden, nämlich
(1) in Typanweisungen,
(2) in IMPLICIT-Anweisungen, und
(3) in FUNCTION-Anweisungen.

Durch eine Typanweisung erhalten zunächst einmal alle Größen, deren Namen aufgeführt sind, die angegebene Länge. So werden durch

```
CHARACTER*15 A,B
CHARACTER*20 T(7,7)
```

zwei Textvariable mit den Namen A und B sowie ein Textfeld mit dem Namen T vereinbart, das aus 49 Elementen besteht. A und B besitzen jeweils die Länge 15; die Elemente von T besitzen jeweils die Länge 20.

In einer Typanweisung besteht daneben die Möglichkeit, auch Größen mit unterschiedlichen Längen zu vereinbaren: Hinter jeden Namen, bei Feldern hinter die Dimensionsbeschreibung, kann eine Längenangabe in der Form *l geschrieben werden. Eine derartige Längenangabe gilt immer nur für die Größe, hinter der sie steht. Die beiden Typanweisungen, die oben angegeben sind, können damit, ohne daß sich am Resultat etwas ändert, zu einer Typanweisung zusammengefaßt werden:

```
CHARACTER*15 A,B,T(7,7)*20
```

Äquivalent sind aber auch

```
CHARACTER A*15,B*15,T(7,7)*20
```

und

```
CHARACTER*20 A*15,B*15,T(7,7)
```

Wie für Felder anderer Datentypen muß auch für Textfelder die Dimensionierung nicht notwendig in einer Typanweisung erfolgen, sondern kann alternativ in einer DIMENSION-Anweisung (siehe 2.4) oder in einer COMMON-Anweisung (siehe 8.1) vorgenommen werden.

Weitere Beispiele:

(1) Die Anweisung

```
IMPLICIT CHARACTER*25 (T)
```

bewirkt, daß alle Namen, die mit dem Buchstaben T beginnen und nicht in einer Typanweisung aufgeführt sind, den Typ CHARACTER und die Länge 25 erhalten.

(2) Die Anweisung

```
CHARACTER*17 FUNCTION T1(Z)
```

legt fest, daß T1 eine Funktion mit einem Parameter ist. Der Funktionswert von T1 ist ein Text mit der Länge 17.

Man beachte: Der Name eines Eingangspunktes eines Funktionsunterprogramms darf dann und nur dann den Typ CHARACTER besitzen, wenn die Funktion selbst den Typ CHARACTER besitzt. Zusätzlich müssen alle Eingangspunkte einer Funktion dieselbe Länge wie die Funktion besitzen, d.h. die Funktion selbst und die Namen aller ihrer Eingangspunkte müssen entweder sämtlich mit der Länge (*) oder sämtlich mit gleichwertigen konstanten INTEGER-Ausdrücken als Länge vereinbart sein.

5.1.2 Textkonstanten

Textkonstanten haben die Form

$'h_1h_2\cdots h_n'$

Hier steht jedes h für ein beliebiges Zeichen (außer dem Apostroph) aus dem Zeichenvorrat des Rechners, muß n positiv sein.
Falls ein Apostroph Bestandteil des Wertes sein soll, müssen an der entsprechenden Stelle in der Konstanten zwei unmittelbar aufeinanderfolgende (nicht durch Leerzeichen getrennte) Apostrophe angege-

ben werden.
Man beachte: Die Apostrophe, mit denen eine Textkonstante beginnt und endet, dienen nur zu ihrer formalen Kennzeichnung; sie sind kein Bestandteil des Wertes, den die Konstante repräsentiert. Dagegen sind Leerzeichen innerhalb einer Textkonstanten, anders als sonst in FORTRAN, signifikante Zeichen.

Beispiele:

Konstante	Wert
'F(X)'	F(X)
'F''(X)'	F'(X)
'A B'	A␣B
'''HEUTE'''	'HEUTE'

Man beachte: Textkonstanten müssen, wie andere Textgrößen, mindestens die Länge 1 besitzen. Eine maximale Länge legt die Norm nur indirekt durch die maximal zulässige Anzahl der Fortsetzungszeilen für eine Anweisung fest. Für einzelne Rechner gibt es engere Grenzen; so läßt VS FORTRAN von IBM 500 Zeichen und FORTRAN (ASCII) von UNIVAC 511 Zeichen wie bei anderen Textgrößen als maximale Länge zu. Teilweise liegt die Obergrenze der Länge für Textkonstanten deutlich unter der für andere Textgrößen; VAX-11 FORTRAN von DEC läßt für Textvariable zwar die Länge 32767 zu, für Textkonstanten jedoch nur die Länge 2000.

5.1.3 Teilketten

Bei arithmetischen und logischen Variablen reicht es i.a. aus, wenn man auf ihre Werte als Ganzes zugreifen kann. Bei Textvariablen wird man dagegen vielfach auch auf Teile der Zeichenfolge zugreifen wollen. Die Möglichkeit hierzu bieten die Teilketten (substrings). Eine Teilkette hat die Form

$$v([e_1]:[e_2])$$

oder

$$a(s[,s]...)([e_1]:[e_2])$$

Hierin sind

v	Name einer Textvariablen
a(s[,s]...)	Name eines Elements eines Textfeldes
e_1 und e_2	INTEGER-Ausdrücke, deren Werte die Position des ersten bzw. letzten Zeichens der Teilkette innerhalb der Variablen bzw. des Feldelements bezeichnen. Wenn l die Länge der Variablen bzw. des Feldelements ist, muß gelten $1 \leq e_1 \leq e_2 \leq l.$

Man beachte: Eine Teilkette besteht immer aus Zeichen, die in der Textvariablen bzw. im Element des Textfeldes unmittelbar aufeinanderfolgen. Beispiel: Es sei vereinbart

```
CHARACTER*6 A,B(10,10)
```

Dann bezeichnet A(2:4) die Zeichenpositionen zwei bis vier in der Variablen A und B(2,6)(3:5) die Zeichenpositionen drei bis fünf im Feldelement B(2,6).

Falls e_1 bzw. e_2 nicht angegeben ist, wird automatisch der Wert 1 bzw. die Länge der Variablen bzw. des Feldelements ergänzt.

Beispiel: Ist wieder vereinbart

```
CHARACTER*6 A,B(10,10)
```

so sind die folgenden Teilketten jeweils paarweise äquivalent:

A(:3)	A(1:3)
B(7,4)(5:)	B(7,4)(5:6)
B(3,1)(:)	B(3,1)(1:6)

Außerdem sind natürlich äquivalent

B(3,1)(:)	B(3,1)

5.2 Arbeiten mit Texten

5.2.1 Textausdrücke

Die einfachsten Textausdrücke sind
(1) (benannte und unbenannte) Textkonstanten,
(2) Textvariable,
(3) Elemente von Textfeldern,
(4) Teilketten, und
(5) Aufrufe von Funktionen, die den Typ CHARACTER besitzen.

Je zwei Textausdrücke können durch den Konkatenationsoperator // zu einem Textausdruck verknüpft werden. Bei einer Konkatenation werden die Zeichenfolgen der beiden Operanden aneinandergefügt.

Zum Beispiel ergibt der Textausdruck 'A'//'*'//'B'//'='//'C' als Resultat die Zeichenfolge A*B=C.

5.2.2 Wertzuweisung

Die Wertzuweisung hat die Form

v=e

Hierin sind

v eine Textvariable, ein Element eines Textfeldes oder eine Teilkette
e ein Textausdruck

Da automatische Umwandlungsroutinen zwischen dem Typ CHARACTER und den anderen Datentypen nicht zur Verfügung stehen, ist eine Wertzuweisung v=e mit einem Textausdruck e dann und nur dann zulässig, wenn v den Typ CHARACTER besitzt.
Allerdings brauchen v und e nicht dieselbe Länge zu besitzen. Ist v länger als e, werden die Zeichen von e linksbündig in v gespeichert und die restlichen Zeichenpositionen von v mit Leerzeichen gefüllt. Ist umgekehrt v kürzer als e, werden, links beginnend, nur so viele Zeichen von e übertragen, bis v gefüllt ist; die restlichen Zeichen von e gehen verloren.

Man beachte: Bei einer Wertzuweisung v=e werden alle Zeichen, die bislang in v standen, überschrieben. Ist allerdings v eine Teilkette, so wird nur der bisherige Wert dieser Teilkette überschrieben, während die Zeichen der Variablen bzw. des Feldelements, die nicht zu der Teilkette gehören, unverändert bleiben.

Beispiel: Die Ausführung der Anweisungsfolge

```
CHARACTER*6 A,B,C,D*3
A='*'
B='ABCDEFG'
C='GAFFEN'
C(3:4)='RT'
D='X+Y'
D(2:2)=A
```

erzeugt in A die Zeichenfolge *␣␣␣␣␣, in B die Zeichenfolge ABCDEF, in C die Zeichenfolge GARTEN und in D die Zeichenfolge X*Y.

5.2.3 Vergleiche von Texten

Zwei Textausdrücke können, wie zwei arithmetische Ausdrücke, durch einen Vergleichsoperator zu einem Vergleichsausdruck verknüpft werden. Der Wert eines Vergleichausdrucks ist ein logischer Wert, also "wahr" oder "falsch". Verwendet werden können alle sechs Vergleichsoperatoren (siehe 3.1).

Vergleiche von Textausdrücken werden prinzipiell zeichenweise von links nach rechts ausgeführt; bei ungleicher Länge der Operanden werden an den kürzeren Leerzeichen angehängt.

Ob ein Zeichen "größer" oder "kleiner" als ein anderes ist, hängt von dem jeweils verwendeten Code ab. In jedem Code wird eine bestimmte Reihenfolge der enthaltenen Zeichen festgelegt, die als Sortierfolge bezeichnet wird. Jedes Zeichen kann dann mit der Nummer der Position, an der es steht, identifiziert werden; ein Zeichen ist "größer" bzw. "kleiner" als ein anderes, wenn die Nummer seiner Position in der Sortierfolge größer bzw. kleiner als die des anderen Zeichens ist.

Die Sortierfolgen verschiedener Codes stimmen i.a. nicht überein. Deshalb sieht FORTRAN für Vergleiche von Textausdrücken zwei Möglichkeiten vor:

(1) Bei Verwendung eines der vier Vergleichsoperatoren .LE., .LT., .GE. und .GT. wird die Sortierfolge des Code verwendet, mit der der Rechner gewöhnlich arbeitet, also die Sortierfolge z.B. des ASCII-Code oder des EBCDIC-Code (siehe Anhang A) oder auch eines anderen Code.

(2) Bei Verwendung einer der vier inneren Standardfunktionen LLE, LLT, LGE und LGT (siehe Anhang B.4) mit den zu vergleichenden Textausdrücken als aktuellen Parametern wird immer die Sortierfolge des ASCII-Code zugrunde gelegt.

Unabhängig von der jeweils verwendeten Sortierfolge ist natürlich die Gleichheit bzw. Ungleichheit von zwei Textausdrücken. So stehen für diese Vergleiche auch nur die beiden Vergleichsoperatoren .EQ. und .NE. zur Verfügung.

Zur Verwendung für FORTRAN läßt die Norm alle Codes zu, die folgenden (Mindest-)Anforderungen genügen:

(1) Die (Groß-)Buchstaben sind alphabetisch aufsteigend sortiert, d.h. es gilt

$$A < B < \ldots\ldots < Z$$

(2) Die Ziffern sind nach ihren Werten aufsteigend sortiert, d.h. es gilt

$$0 < 1 < \ldots\ldots < 9$$

(3) Das Leerzeichen ist kleiner als jeder Buchstabe und jede Ziffer, d.h. es gilt $\not b < A$ und $\not b < 0$.

(4) Ziffern und (Groß-)Buchstaben sind nicht vermischt, d.h. es gilt entweder $9 < A$ oder $Z < 0$.

(5) Die Sonderzeichen des FORTRAN-Zeichensatzes können in der Sortierfolge an beliebiger Stelle stehen, also insbesondere auch eingestreut zwischen die Buchstaben und Ziffern.

Beispiele:

Ausdruck	Wert
'X'.EQ.'X '	"wahr"
'X'.EQ.' X'	"falsch"
'DIENSTAG'.LE.'MITTWOCH'	"wahr"
'05'.GT.'01'	"wahr"
'05'.GT.' 5'	"wahr"
LLE('A+B','A*B')	"falsch"
'A+B'.LE.'A*B'	maschinenabhängig entweder "wahr" (z.B. EBCDIC) oder "falsch" (z.B. ASCII)

Durch die Vorschriften der Norm wird sichergestellt, daß Texte, die nur aus Ziffern, (Groß-)Buchstaben und Leerzeichen bestehen, in einem FORTRAN-Programm problemlos lexikographisch sortiert werden können. Man beachte jedoch, daß dieses bereits für Texte, die gemischt aus Klein- und Großbuchstaben bestehen, nicht mehr der Fall ist. So besitzt z.B. der Vergleichsausdruck 'A'.NE.'a' immer den Wert "wahr". Vor einem Vergleich von Texten ist man also, wenn man eine alphabetische Sortierung wünscht, i.a. gezwungen, zunächst Kleinbuchstaben in Großbuchstaben umzuwandeln. Ein Unterprogramm, das derartige Umwandlungen ausführt, ist als Beispiel in Abschnitt 5.2.5 angegeben.

5.2.4 Die Länge (*)

In drei Fällen kann die Länge einer Textgröße durch die Angabe (*) vereinbart werden:

(1) Wenn der Name einer benannten Konstanten (siehe 2.1.4) mit der Länge (*) vereinbart wird, bestimmt der Rechner die tatsächliche Länge aus der zugehörigen PARAMETER-Anweisung.

(2) Wenn ein formaler Parameter eines externen Unterprogramms mit der Länge (*) vereinbart ist, wird bei jedem Aufruf des Unterprogramms die Länge des entsprechenden aktuellen Parameters verwendet.

(3) Wenn in einer externen Funktion der Name der Funktion und die Namen eventueller Eingangspunkte mit der Länge (*) vereinbart sind, wird bei jedem Aufruf die Länge verwendet, mit der der Funktionsname bzw. der Name des jeweiligen Eingangspunktes im rufenden Programm vereinbart ist.

Ausdrücke, bei denen mindestens ein Operand, der keine benannte Konstante ist, mit der Länge (*) vereinbart ist, unterliegen gewissen Beschränkungen in der Verwendung: Sie dürfen nicht angegeben werden als
(1) aktuelle Parameter in einem Unterprogramm-Aufruf, nicht als
(2) aktuelle Parameter von Anweisungsfunktionen, nicht als
(3) Elemente einer Ausgabeliste, und nicht als
(4) Format in einem Steuerparameter [FMT=]f (siehe 7.1).
Die Größen, die mit der Länge (*) vereinbart sind, unterliegen selbst diesen Beschränkungen nicht.

Man beachte: Bei Anweisungsfunktionen dürfen weder der Funktionsname selbst noch einer der formalen Parameter mit der Länge (*) vereinbart werden. Ebenso darf weder der Name einer externen Funktion noch der Name eines Eingangspunktes einer externen Funktion in der rufenden Programmeinheit mit der Länge (*) vereinbart werden.

5.2.5 Beispiele

(1) Zur Umwandlung von Groß- in Kleinbuchstaben bzw. umgekehrt dient das Unterprogramm GROSSB mit dem zusätzlichen Eingangspunkt KLEINB:

```
      CHARACTER*(*) FUNCTION GROSSB(T)
      CHARACTER*(*) KLEINB, T, TAB(2)*26
      TAB(1)='abcdefghijklmnopqrstuvwxyz'
      TAB(2)='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      GOTO 1
      ENTRY KLEINB(T)
      TAB(1)='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      TAB(2)='abcdefghijklmnopqrstuvwxyz'
    1 DO 2 L=1,LEN(T),1
```

```
      J=INDEX(TAB(1),T(L:L))
      IF (J.EQ.0) THEN
         GROSSB(L:L)=T(L:L)
      ELSE
         GROSSB(L:L)=TAB(2)(J:J)
      END IF
    2 CONTINUE
      END
```

Die hier verwendeten Funktionen LEN und INDEX sind innere Standardfunktionen (siehe Anhang B.4).

Mit diesem Unterprogramm liefert die Anweisungsfolge

```
      CHARACTER*6 A,B,C,GROSSB*500,KLEINB*500
      C='A*B=ab'
      A=GROSSB(C)
      B=KLEINB(C)
```

in A die Zeichenfolge A*B=AB und in B die Zeichenfolge a*b=ab. Sonderzeichen werden durch das Unterprogramm nicht verändert.

Anmerkung: In dieser Anweisungsfolge hätte es natürlich ausgereicht, nicht nur die Variablen A und B, sondern auch die Funktionen GROSSB und KLEINB mit der Länge 6 zu vereinbaren. Die größere Länge von GROSSB und KLEINB läßt allerdings die Möglichkeit offen, in der rufenden Programmeinheit an anderer Stelle auch längere Funktionswerte zu erhalten, nämlich maximal bis zu 500 Zeichen.

In der Praxis wird man das Unterprogramm GROSSB etwas anders formulieren, nämlich unter Verwendung der Anweisung DATA (siehe 8.3), dafür ohne Sprunganweisung und ohne Wertzuweisungen für die Elemente des Feldes TAB. Das Unterprogramm kann etwa lauten:

```
      CHARACTER*(*) FUNCTION GROSSB(T)
      CHARACTER*(*) KLEINB, T, TAB(2)*26
      DATA TAB/'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
     *         'abcdefghijklmnopqrstuvwxyz'/,
     *     I/1/
```

```
      I=2
      ENTRY KLEINB(T)
      DO 2 L=1,LEN(T),1
      J=INDEX(TAB(I),T(L:L))
      IF (J.EQ.0) THEN
         GROSSB(L:L)=T(L:L)
      ELSE
         GROSSB(L:L)=TAB(3-I)(J:J)
      END IF
    2 CONTINUE
      I=1
      END
```

Anmerkung: Die Wertzuweisung für I unmittelbar vor dem Rücksprung ist eingefügt, um sicherzustellen, daß das Unterprogramm auf allen Rechnern einwandfrei läuft. Die Hintergründe hierfür werden im Zusammenhang mit der SAVE-Anweisung erläutert (siehe 8.5).

(2) Bei einer Konkatenation werden die Zeichenfolgen der Operanden grundsätzlich in voller Länge aneinandergefügt, also einschließlich aller enthaltenen Leerzeichen. Die Anweisungsfolge

```
CHARACTER T1*5,T2*5,T3*10
T1='X'
T2='Y'
T3=T1//'*'//T2
```

liefert so in T3 die Zeichenfolge X␣␣␣␣*Y␣␣␣␣.
Will man bei einer Konkatenation nur die wesentlichen Teile der Zeichenfolgen, also ohne die eventuell am Ende stehenden Leerzeichen, aneinanderfügen, so muß man die entsprechenden Teilketten als Operanden der Konkatenation angeben. Das folgende Unterprogramm TRIMLE bestimmt für einen Textausdruck die Länge des wesentlichen Teils der Zeichenfolge:

```
      INTEGER FUNCTION TRIMLE(T)
      CHARACTER T*(*)
      DO 1 TRIMLE=LEN(T),1,-1
    1 IF (T(TRIMLE:TRIMLE).NE.' ') RETURN
      TRIMLE=1
      END
```

Man beachte, daß dieses Unterprogramm den Wert 1 liefert, wenn sein aktueller Parameter nur Leerzeichen enthält.
Mit dem Unterprogramm TRIMLE liefert die Anweisungsfolge

```
      CHARACTER T1*5,T2*5,T3*10
      INTEGER TRIMLE
      T1='X'
      T2='Y'
      T3=T1(1:TRIMLE(T1))//'*'//T2(1:TRIMLE(T2))
```

in T3 die Zeichenfolge X*Y␣␣␣␣␣␣␣␣.

5.2.6 Texte als Parameter von Unterprogrammen

Bei (arithmetischen und logischen) Feldern, die als aktuelle Parameter an ein Unterprogramm übergeben werden, braucht die Dimensionierung nicht mit der Dimensionierung des entsprechenden formalen Parameters übereinzustimmen (siehe 4.3.2). Das gleiche gilt auch für Textfelder.

Ähnlich braucht, wenn es sich um Textgrößen handelt, die Länge eines aktuellen Parameters nicht mit der Länge des entsprechenden formalen Parameters übereinzustimmen, solange nur sichergestellt ist, daß alle Zeichenpositionen, auf die das Unterprogramm mittels des formalen Parameters zugreift, zu dem Speicherbereich gehören, den der jeweilige aktuelle Parameter belegt. Für die Praxis heißt das, daß ein aktueller Parameter dann und nur dann angegeben werden darf, wenn er die gleiche oder eine größere Länge als der entsprechende formale Parameter besitzt.

Die Länge (*) bewirkt für einen formalen Parameter gerade, daß er immer die gleiche Länge hat wie der entsprechende aktuelle Parame-

ter. Ist dagegen ein formaler Parameter kürzer als der entsprechende aktuelle Parameter, so wird im Unterprogramm nur auf die links im aktuellen Parameter stehende Teilkette, und zwar in der Länge des formalen Parameters, zugegriffen.

Darüber hinaus ist es bei Textfeldern, die als aktuelle Parameter angegeben werden, auch möglich, den gesamten Inhalt des Feldes oder beliebige (aufeinanderfolgende) Teile davon im Unterprogramm als eine zusammenhängende Zeichenfolge zu betrachten. Natürlich können die Inhalte verschiedener Elemente eines Textfeldes nur dann als zusammenhängende Zeichenfolge betrachtet werden, wenn diese Elemente im Speicher unmittelbar aufeinanderfolgen.

Beispiel: Gegeben sei das Unterprogramm

```
      SUBROUTINE ANZAHL(T,L,TAB)
      CHARACTER T*(*),ZEI*63
      INTEGER TAB(0:3)
      ZEI(1:11)=' 0123456789'
      ZEI(12:37)='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      ZEI(38:63)='abcdefghijklmnopqrstuvwxyz'
      DO 1 I=0,3,1
    1 TAB(I)=0
      DO 2 I=1,L,1
      J=INDEX(ZEI,T(I:I))
      IF (J.GT.1) THEN
         IF (J.LE.11) THEN
            J=2
         ELSE
            J=3
         END IF
      END IF
    2 TAB(J)=TAB(J)+1
      END
```

Dieses Unterprogramm zählt, wie oft in dem Text T mit der Länge L Leerzeichen, andere Sonderzeichen, (Klein- und Groß-)Buchstaben sowie Ziffern vorkommen.
Obwohl der formale Parameter T als Textvariable und nicht als Text-

feld vereinbart ist, kann das Unterprogramm nicht nur für Textausdrücke, sondern auch für Textfelder angewandt werden. So schreibt etwa die Anweisungsfolge

```
      CHARACTER TEXT(5,5)*20
      INTEGER ZAHLEN(4)
      TEXT(1,1)='+- 12ABCDEFGHIJKLMNN'
      DO 1 I=1,5,1
      DO 1 J=1,5,1
    1 TEXT(I,J)=TEXT(1,1)
      CALL ANZAHL(TEXT,500,ZAHLEN)
      PRINT *, ZAHLEN
      PRINT *
      CALL ANZAHL(TEXT(1,3),200,ZAHLEN)
      PRINT *, ZAHLEN
```

listengesteuert die drei Zeilen

```
          50          25          50          375

          20          10          20          150
```

Durch den ersten Aufruf wird also die Zählung für die ganze Matrix TEXT, durch den zweiten Aufruf nur für ihre dritte und vierte Spalte ausgeführt.

6 Arbeiten mit Dateien

6.1 Grundbegriffe

6.1.1 Sätze

Ein (Daten-)Satz (record) ist eine Folge von Daten, die in gewissem Sinne zusammengehören, z.B. der Inhalt einer Lochkarte oder der Inhalt einer Zeile eines Zeilendruckers.

Ein Satz wird als formatierter Satz bezeichnet, wenn er aus Daten in einer externen Darstellung besteht, und als unformatierter Satz, wenn er aus Daten in einer internen Darstellung besteht. Die Übertragung eines Satzes aus dem Hauptspeicher in einen externen Speicher oder umgekehrt aus einem externen Speicher in den Hauptspeicher wird entsprechend als formatierte bzw. unformatierte Übertragung oder Ein-/Ausgabe bezeichnet.

Ob ein Satz, der ausgegeben werden soll, formatiert sein muß oder unformatiert sein darf, hängt nicht zuletzt von dem externen Speicher ab, in den er übertragen werden soll: Sätze, die auf einen Zeilendrucker oder ein Bildschirmgerät ausgegeben werden sollen, müssen z.B. immer formatiert sein; dagegen hat der Programmierer bei Magnetspeichern aller Art i.a. die Wahl zwischen formatierter und unformatierter Ausgabe.
Umgekehrt kann ein Satz natürlich immer nur so gelesen werden, wie er geschrieben wurde. Ein formatiert geschriebener Satz kann also nur formatiert und ein unformatiert geschriebener Satz nur unformatiert gelesen werden.

Die Länge eines formatierten Satzes ist die Anzahl der enthaltenen Zeichen; die Länge eines unformatierten Satzes ist die Anzahl der Worte, die zur Aufnahme der enthaltenen Werte benötigt werden. Wie lang ein Satz sein kann, hängt von dem jeweiligen externen Speicher ab. So wird z.B. durch die Länge der Zeile eines Zeilendruckers festgelegt, wie lang ein (formatierter) Satz höchstens sein darf, der auf diesen Drucker ausgegeben werden soll. Bei Magnetspeichern aller Art gibt es dagegen i.a. keine Beschränkungen für die Länge eines Satzes.

Sowohl ein formatierter als auch ein unformatierter Satz kann leer sein; er besitzt dann die Länge Null. Daneben gibt es spezielle Sätze, die weder formatiert noch unformatiert sind; diese Sätze besitzen keine Länge.

Die Umwandlungen zwischen interner und externer Darstellung, die bei formatierter Ein-/Ausgabe erforderlich werden, werden im Rahmen der jeweiligen Ein-/Ausgabeanweisung ausgeführt. Für den Programmierer gibt es hier zwei Möglichkeiten:

(1) Bei der formatgesteuerten Ein-/Ausgabe muß er selbst detailliert die vorzunehmenden Umwandlungen beschreiben (siehe 7.1).

(2) Bei der listengesteuerten Ein-/Ausgabe wählt der Rechner aufgrund von Standard-Vorgaben anhand der Typen der Elemente der Ein-/Ausgabeliste die vorzunehmenden Umwandlungen aus (siehe 7.2).

6.1.2 Dateien

Eine Datei (file) ist eine Menge von Sätzen, die in gewissem Sinne in sich abgeschlossen ist. Z.B. bilden die Daten, die ein Programm während seiner Ausführung liest, eine Datei.
Eine Datei kann leer sein, also keinen Satz enthalten. So ist z.B. jede Datei leer, bevor ihr erster Satz geschrieben wird.

FORTRAN unterscheidet zwischen internen und externen Dateien: Eine interne Datei besteht aus einer Textvariablen, einem Textfeld oder einem Element eines Textfeldes (siehe 7.3). Eine Datei, die keine interne Datei ist, ist eine externe Datei (eine "Datei außerhalb des Programms").
Externe Dateien werden vom Betriebssystem der Rechenanlage verwaltet. Sie werden i.a. durch Dateinamen gekennzeichnet; für den Aufbau dieser Dateinamen gelten anlagenspezifische Regeln, die sich zumeist von den Regeln für den Aufbau von FORTRAN-Namen unterscheiden.

Die Norm schreibt vor, daß jede Datei entweder nur formatierte oder nur unformatierte Sätze enthalten darf. Man kann deshalb kurz von

formatierten bzw. unformatierten Dateien sprechen.
Interne Dateien sind immer formatiert. Ebenso sind gewisse externe Dateien immer formatiert, z.B. Dateien, die zur Ausgabe auf ein Bildschirmgerät oder einen Zeilendrucker bestimmt sind; bei anderen externen Dateien kann der Programmierer vor dem (erstmaligen) Beschreiben festlegen, ob die Datei formatiert oder unformatiert sein soll.

Wenn in der Folge kurz der Begriff "Datei" verwendet wird, soll damit immer eine externe Datei gemeint sein.

Teilweise werden auch die Begriffe Eingabedatei und Ausgabedatei verwendet, um anzudeuten, daß aus einer Datei nur gelesen bzw. in eine Datei nur geschrieben werden kann.

6.1.3 Dateinummern

Interne Dateien können in FORTRAN-Programmen direkt über ihren Namen angesprochen werden. Bei externen Dateien ist das grundsätzlich nicht möglich; vielmehr muß einer externen Datei zunächst eine Dateinummer (unit identifier) zugeordnet werden. Dateinummern sind nichtnegative ganze Zahlen.

Die Zuordnung zwischen einer Datei und einer Dateinummer kann aus dem Programm heraus erfolgen. Für benannte Dateien läßt die Norm auch zu, daß Zuordnungen bereits vorab vorgenommen werden, d.h. bevor die Ausführung des Programms beginnt, das auf die Datei zugreift; wie derartige Zuordnungen vorzunehmen sind, hängt natürlich von der jeweiligen Rechenanlage ab.
Bei den meisten Rechenanlagen werden, ohne daß die Norm das vorschreibt, mindestens zwei Dateinummern automatisch vorab zugeordnet, nämlich

5 : (Standard-)Eingabedatei
6 : (Standard-)Ausgabedatei

Eine Zuordnung kann im Verlauf eines Programms wieder gelöst werden. Danach kann einer benannten Datei eine andere Dateinummer und einer Dateinummer eine andere Datei zugeordnet werden.

Einer Datei kann eine Dateinummer bereits dann zugeordnet werden, wenn die Datei selbst noch nicht existiert. Bei einem Zugriff auf eine Datei, die noch nicht existiert, kann folgendes passieren:

(1) Die Operation endet mit einem Fehler. Das ist z.B. dann der Fall, wenn ein Satz einer Datei, die noch nicht existiert, gelesen werden soll.

(2) Die Datei wird neu eingerichtet. Das ist z.B. dann der Fall, wenn ein Satz einer Datei, die noch nicht existiert, geschrieben werden soll.

(3) Die Operation liefert als Resultat die Information, daß die Datei nicht existiert.

6.1.4 Zugriff und Positionierung

Wenn ein Satz einer Datei gelesen oder geschrieben werden soll, muß die Datei geeignet positioniert sein. Man kann sich das so vorstellen, daß am Beginn der Übertragung ein Zeiger gesetzt sein muß, der auf den Anfang des zu übertragenden Satzes verweist. Mit der Übertragung ändert sich die Positionierung; nach ihrem Abschluß ist die Datei unmittelbar hinter dem übertragenen Satz positioniert.

Bei Dateien mit sequentiellem Zugriff (sequential file), kurz: sequentielle Datei, braucht der Programmierer sich nicht selbst um die Positionierung zu kümmern. Bevor die erste Datenübertragung erfolgt, wird die Datei automatisch auf ihren Anfang positioniert, d.h. auf den Anfang ihres ersten Satzes. Nach jeder Übertragung wird, wie beschrieben, die Datei automatisch hinter den soeben übertragenen Satz positioniert, und damit auf den Beginn des nächsten Satzes. Praktisch wird also beim Schreiben einer sequentiellen Datei eine Reihenfolge ihrer Sätze festgelegt; ein erneutes Lesen der Sätze kann nur in dieser Reihenfolge erfolgen. Daneben besteht für sequentielle Dateien, unter gewissen Voraussetzungen, nur die Möglichkeit, sie wieder auf den Anfang ihres ersten Satzes zu positionieren, oder auch auf den Anfang des Satzes, hinter dem sie momentan positioniert ist, d.h. um einen Satz "zurückzusetzen" (siehe

6.5).

Anders ist das bei Dateien mit direktem Zugriff (direct file), kurz: direkte Datei. Die Sätze von direkten Dateien werden mit Satznummern, positiven ganzen Zahlen, identifiziert. Die Satznummer, die der Programmierer bei jeder Anweisung zur Datenübertragung anzugeben hat, bewirkt eine entsprechende Positionierung vor dem Beginn der Übertragung. Die Sätze einer direkten Datei können entsprechend in beliebiger Reihenfolge gelesen und geschrieben werden, z.B. erst Satz 3, dann Satz 15, anschließend Satz 1, usw.. Ein Satz, der einmal geschrieben worden ist, kann nicht wieder vollständig gelöscht, sondern nur durch einen anderen Satz überschrieben werden. Ebenso kann die Nummer eines Satzes, nachdem sie einmal festgelegt worden ist, direkt nicht mehr verändert werden. Natürlich besteht die Möglichkeit, Änderungen indirekt vorzunehmen: Soll z.B. der Satz, der bislang die Nummer 17 hatte, die Nummer 9 erhalten, so kann man ihn zunächst lesen und dann mit der (neuen) Nummer 9 wieder schreiben. Da hierbei der Satz 17 seinen alten Inhalt behält, muß man ihn ggf. noch überschreiben, z.B. mit Leerzeichen, wenn er formatiert ist, oder mit Nullen, wenn er unformatiert ist; vollständig gelöscht werden kann er nicht.

Für direkte Dateien sieht die Norm gewisse Einschränkungen vor:

(1) Alle Sätze einer direkten Datei müssen die gleiche Länge besitzen.
(2) Für formatierte direkte Dateien ist nur formatgesteuerte, nicht jedoch listengesteuerte Ein-/Ausgabe zulässig (siehe 7.1).
(3) Ein Satz darf erst gelesen werden, nachdem er (vom gleichen oder auch einem anderen Programm) geschrieben wurde.

Die Organisationsform einer Datei läßt sich, nachdem sie einmal festgelegt ist, nicht mehr ändern; so kann eine Datei, die sequentiell beschrieben wurde, nur wieder sequentiell gelesen werden. Einzige Ausnahme: Eine Datei, die als direkte Datei geschrieben wurde, kann sequentiell gelesen werden; die Sätze werden dann in der Reihenfolge ihrer Satznummern geliefert, und zwar in der Reihenfolge 1, 2, 3, ... unabhängig davon, ob Sätze mit diesen Nummern geschrieben worden waren oder nicht. Da auch hier die Einschränkung

gilt, daß nur Sätze gelesen werden dürfen, die zuvor geschrieben wurden, können nur "lückenlos" beschriebene direkte Dateien sequentiell gelesen werden.

Ob eine Datei direkt sein kann oder sequentiell sein muß, hängt sowohl von dem Speichermedium ab, auf dem sie sich befindet, als auch von ihrer geplanten Verwendung. So sind z.B. Dateien auf Lochkarten immer sequentiell; ebenso müssen Dateien, die zur Ausgabe auf ein Bildschirmgerät oder einen Zeilendrucker bestimmt sind, sequentiellen Zugriff erlauben; bei Dateien auf Magnetplatten oder Magnettrommeln, die durch ein (FORTRAN-)Programm geschrieben und auch wieder gelesen werden sollen, kann der Programmierer dagegen i.a. wählen, wie die Datei organisiert sein soll.

6.1.5 Dateiende

Sequentielle Dateien können hinter ihrem letzten Datensatz noch einen speziellen Satz zur Kennzeichnung des Dateiendes enthalten, den EOF-Satz (End Of File). Dieser Satz ist weder formatiert noch unformatiert, besitzt entsprechend keine Länge.

Beim Erstellen einer Datei muß der Programmierer i.a. selbst durch eine spezielle Anweisung dafür sorgen, daß ein EOF-Satz geschrieben wird. Beim Lesen einer Datei läßt sich ein EOF-Satz vom Programm aus als solcher erkennen, so daß dann im Programm entsprechend reagiert werden kann (siehe 6.2.4).

6.2 Die Anweisungen

FORTRAN kennt drei Gruppen von Ein-/Ausgabeanweisungen:

(1) Zur Datenübertragung dienen die Anweisungen

READ	(siehe 6.4)
WRITE	(siehe 6.4)
PRINT	(siehe 6.4)

(2) Zur Positionierung sequentieller Dateien dienen die Anweisungen

BACKSPACE	(siehe 6.5)
REWIND	(siehe 6.5)
ENDFILE	(siehe 6.5)

(3) Zur Dateiverwaltung dienen die Anweisungen

OPEN	(siehe 6.3)
CLOSE	(siehe 6.6)
INQUIRE	(siehe 6.7)

Viele FORTRAN-Versionen kennen zusätzlich die von der Norm nicht vorgeschriebene Anweisung

PUNCH	(siehe 6.5)

die ebenfalls zur Datenübertragung dient.

6.2.1 Aufbau der Anweisungen

Die Anweisungen können in der allgemeinen Form

name(stliste) [ealiste]

und/oder in der Kurzform

name stpar [,ealiste]

geschrieben werden. Hierin sind:

name	Schlüsselwort zur Bezeichnung der Anweisung
stliste	Liste von Steuerparametern
stpar	einzelner Steuerparameter
ealiste	Ein-/Ausgabeliste

Die allgemeine Form steht für alle Anweisungen außer PRINT (und PUNCH), die Kurzform für alle Anweisungen außer WRITE, OPEN, CLOSE und INQUIRE zur Verfügung.

Eine Ein-/Ausgabeliste kann bei allen Anweisungen zur Datenübertragung angegeben werden; bei allen anderen Anweisungen entfällt sie.

Zur Verfügung stehen insgesamt 22 verschiedene Steuerparameter. Von diesen sollen in diesem Abschnitt 4 erläutert werden, die restlichen 18 später im Zusammenhang mit den einzelnen Anweisungen. Eini-

ge Steuerparameter besitzen Kurzformen, bei deren Verwendung der Programmierer bestimmte Reihenfolgen einhalten muß; sonst können Steuerparameter in beliebiger Reihenfolge angegeben werden. In jeder Liste von Steuerparametern darf jeder Steuerparameter, der für die Anweisung überhaupt zulässig ist, höchstens einmal stehen.

6.2.2 Der Steuerparameter UNIT

Bei den Ein-/Ausgabeanweisungen muß zumeist explizit angegeben werden, auf welche Datei zugegriffen werden soll. Diese Angabe erfolgt durch den Steuerparameter

[UNIT=]u

Hier bestehen für u folgende Möglichkeiten:

(1) Bei allen Anweisungen, bei denen überhaupt eine Dateinummer anzugeben ist, kann u die Dateinummer einer externen Datei in der Form eines INTEGER-Ausdrucks sein.

(2) Nur bei READ- und WRITE-Anweisungen in der allgemeinen Form kann u ein Stern sein. Damit wird gekennzeichnet, daß aus der Standard-Eingabedatei gelesen bzw. in die Standard-Ausgabedatei geschrieben werden soll.

(3) Ebenfalls nur bei READ- und WRITE-Anweisungen in der allgemeinen Form kann u der Name einer internen Datei sein (siehe 7.3).

Falls das Schlüsselwort UNIT= nicht angegeben wird, muß u an erster Stelle in der Liste stehen.

Ausnahmen sind die Kurzform der Anweisung READ sowie die Anweisungen PRINT und, soweit verfügbar, PUNCH. Bei ihnen ist die Dateinummer nicht explizit anzugeben, sondern steckt bereits implizit im Namen der Anweisung: READ liest aus der Standard-Eingabedatei, PRINT schreibt in die Standard-Ausgabedatei und PUNCH schreibt in die Standard-Stanzdatei.

6.2.3 Behandlung von Fehlern

Zwei Steuerparameter dienen zur Behandlung von Fehlern, die während einer Ein-/Ausgabeoperation auftreten können, nämlich

ERR=s

mit einer Marke s, sowie

IOSTAT=ios

mit einer Variablen oder einem Feldelement ios vom Typ INTEGER.

Bei Beendigung einer Ein-/Ausgabeoperation ohne Fehler hat der Parameter ERR=s keine Wirkung, erhält ios den Wert Null. Tritt dagegen während einer Ein-/Ausgabeoperation ein Fehler auf, passiert folgendes:

(1) Die Ein-/Ausgabeoperation wird beendet. Falls es sich um eine Operation handelt, die die Positionierung der Datei beeinflußt, wird die Positionierung der Datei unbestimmt; falls durch die Anweisung gelesen werden sollte, werden die Werte aller Elemente der Eingabeliste undefiniert.

(2) Falls keiner der beiden Parameter angegeben ist, wird das Programm insgesamt beendet.

(3) Falls der Parameter IOSTAT=ios angegeben ist, erhält ios einen positiven Wert. Die Norm legt diesen Wert nicht weiter fest; unterschiedliche Fehler können also anlagenabhängig durch unterschiedliche Werte gekennzeichnet werden.

(4) Falls der Parameter ERR=s angegeben ist, erfolgt ein Sprung zu der Marke s; sonst wird das Programm normal fortgesetzt.

Falls in einer Ein-/Ausgabeanweisung nur der Parameter IOSTAT=ios angegeben ist und nicht der Parameter ERR=s, ist der Programmierer i.a. gezwungen, unmittelbar nach der Anweisung anhand des Wertes von ios zu überprüfen, ob die Anweisung fehlerfrei beendet wurde oder nicht.

6.2.4 Erkennen des Dateiendes

Zwei Steuerparameter ermöglichen es, aus dem Programm heraus zu erkennen, ob bei einer sequentiellen Eingabeoperation der EOF-Satz der Datei gelesen wurde, nämlich

END=s

mit einer Marke s, sowie erneut

IOSTAT=ios

mit einer Variablen oder einem Feldelement ios vom Typ INTEGER.

Falls bei einer Eingabeoperation nicht der EOF-Satz gelesen wurde, hat der Parameter END=s keine Wirkung, erhält ios den Wert Null. Wird dagegen erkannt, daß der EOF-Satz der Datei gelesen wurde, passiert folgendes:

(1) Die Eingabeoperation wird beendet. Die Werte aller Elemente der Eingabeliste werden undefiniert.

(2) Falls keiner der beiden Parameter angegeben ist, wird das Programm insgesamt beendet.

(3) Falls der Parameter IOSTAT=ios angegeben ist, erhält ios einen negativen Wert.

(4) Falls der Parameter END=s angegeben ist, erfolgt ein Sprung zu der Marke s; sonst wird das Programm normal fortgesetzt.

Falls in einer Eingabeanweisung nur der Parameter IOSTAT=ios und nicht der Parameter END=s angegeben ist, ist der Programmierer i.a. gezwungen, unmittelbar nach der Anweisung anhand des Wertes von ios zu überprüfen, ob durch die Anweisung ein EOF-Satz gelesen wurde oder nicht.

6.2.5 Ein-/Ausgabelisten

Elemente von Ein-/Ausgabelisten können sein:
(1) Variable
(2) Feldelemente

(3) Teilketten

(4) Felder

Hier greift die Operation nacheinander auf alle Feldelemente in der Reihenfolge zu, in der sie im Speicher aufeinanderfolgen. Die tatsächliche Größe eines Feldes muß der jeweiligen Programmeinheit bekannt sein, d.h. in Unterprogrammen dürfen in einer Ein-/Ausgabeliste nur Felder angegeben werden, bei denen in der Vereinbarung die obere Grenze für den Index der letzten Dimension nicht durch einen Stern beschrieben wird.

(5) Implizite DO-Elemente der Form

(doliste,i=e_1,e_2[,e_3])

Hierin ist doliste eine Ein-/Ausgabeliste.

i, e_1, e_2 und e_3 haben dieselbe Bedeutung und unterliegen denselben Regeln wie bei der DO-Anweisung. Nur wird hier für jeden Wert der Laufvariablen i nicht eine Folge von Anweisungen durchlaufen, sondern eine Folge von Listenelementen erzeugt. Man beachte: Eine doliste darf ihrerseits implizite DO-Elemente enthalten. Implizite DO-Elemente, die in dieser Weise geschachtelt sind, werden von innen nach außen abgearbeitet.

Zusätzlich dürfen Ausgabelisten, nicht jedoch Eingabelisten, als Elemente enthalten:

(6) Ausdrücke beliebiger Art mit zwei Ausnahmen:

- Bei Konkatenationen dürfen Operanden nur dann mit der Länge (*) vereinbart sein, wenn es sich bei ihnen um benannte Konstanten handelt.
- Funktionsunterprogramme, die selbst Ausgabeoperationen vornehmen, dürfen nicht aufgerufen werden.

Beispiel: In einem Programm sei vereinbart

```
DIMENSION A(3,2)
```

Dann wird durch die drei Ein-/Ausgabelisten

```
A
A(1,1),A(2,1),A(3,1),A(1,2),A(2,2),A(3,2)
((A(I,J),I=1,3),J=1,2)
```

jeweils in der gleichen Reihenfolge (spaltenweise) auf alle Elemen-

te des Feldes A zugegriffen. Zeilenweise wird dagegen durch die beiden Ein-/Ausgabelisten

```
A(1,1),A(1,2),A(2,1),A(2,2),A(3,1),A(3,2)
((A(I,J),J=1,2),I=1,3)
```

auf alle Elemente des Feldes A zugegriffen.

6.3 OPEN

Durch die Anweisung

```
OPEN(stliste)
```

kann aus einem Programm heraus die Zuordnung zwischen einer Datei und einer Dateinummer vorgenommen werden. Außerdem können bzw. müssen durch diese Anweisung die Eigenschaften der Datei festgelegt werden.

Immer anzugeben ist der Steuerparameter

```
[UNIT=]u            (siehe 6.2.2)
```

Zusätzlich können, je nach Bedarf, die folgenden Steuerparameter in beliebiger Kombination angegeben werden:

```
IOSTAT=ios          (siehe 6.2.3)
ERR=s               (siehe 6.2.3)
FILE=fn
STATUS=sta
ACCESS=acc
FORM=fm
RECL=rl
BLANK=blnk
```

6.3.1 Der Steuerparameter FILE

Im Steuerparameter FILE=fn bezeichnet fn den Namen der Datei, der die Dateinummer u zuzuordnen ist. fn muß ein Textausdruck sein, der linksbündig den (nach anlagenspezifischen Regeln aufgebauten) Namen der Datei enthält. Dem Namen dürfen Leerzeichen beliebig folgen.

Wenn der Steuerparameter FILE=fn angegeben ist, bestehen mehrere Möglichkeiten:

(1) Falls die Datei fn und die Dateinummer u einander bereits durch eine OPEN-Anweisung zugeordnet sind, müssen alle Steuerparameter, die Eigenschaften der Datei beschreiben, außer BLANK=blnk, dieselben Werte wie zuvor besitzen. Ein neuer Wert von blnk ersetzt den alten Wert.

(2) Unzulässig ist es, einer Datei, der bereits eine Dateinummer zugeordnet ist, eine zweite Dateinummer zuzuordnen.

(3) Falls die Dateinummer u bereits einer anderen Datei zugeordnet ist, wird diese Zuordnung zunächst durch eine (automatische) CLOSE-Anweisung gelöst (siehe 6.6). Anschließend wird die neue Zuordnung wie in einem der folgenden Fälle vorgenommen.

(4) Falls eine Datei fn noch nicht existiert, wird zunächst die Zuordnung zwischen ihr und der Dateinummer u vorgenommen, sofern diese Zuordnung nicht bereits vorab erfolgt war.
Außerdem werden, unabhängig davon, ob die Zuordnung vorab oder durch die OPEN-Anweisung hergestellt wurde, die Eigenschaften der neuen Datei fn anhand der weiteren Steuerparameter festgelegt. Man beachte: Zuordnungen zwischen Dateien und Dateinummern außerhalb eines Programms legen, anders als Zuordnungen durch OPEN-Anweisungen, keine Eigenschaften einer Datei fest.

(5) Falls eine Datei fn bereits existiert, wird nur die Zuordnung zwischen ihr und der Dateinummer u vorgenommen. Bei den restlichen Steuerparametern wird geprüft, ob sie für die Datei fn zulässig sind.

Wenn der Steuerparameter FILE=fn nicht angegeben ist, bestehen ebenfalls mehrere Möglichkeiten:

(6) Falls die Dateinummer u durch eine vorhergehende OPEN-Anweisung einer Datei zugeordnet wurde, wird wie im Fall (1) verfahren.

(7) Falls die Dateinummer u außerhalb des Programms vorab einer Datei zugeordnet wurde, wird wie im Fall (4) bzw. (5) verfahren.

(8) Falls die Dateinummer u noch keiner Datei zugeordnet wurde,

wird eine neue, unbenannte Datei eingerichtet. Diese erhält die Eigenschaften, die durch die restlichen Parameter festgelegt werden.

Eine einmal vorgenommene Zuordnung gilt für alle Programmeinheiten des ausführbaren Programms und bleibt so lange bestehen, bis eine entsprechende CLOSE-Anweisung (siehe 6.6) ausgeführt wird. Zuordnungen, die durch OPEN-Anweisungen vorgenommen wurden, enden jedoch spätestens mit der Beendigung des Programms.

6.3.2 Der Steuerparameter STATUS

Der Steuerparameter STATUS=sta beschreibt die Existenz einer Datei. sta ist ein Textausdruck, der, abgesehen von zusätzlichen, nachfolgenden Leerzeichen, linksbündig einen der folgenden Werte besitzen muß:

OLD — Die Datei existiert bereits.

NEW — Die Datei existiert noch nicht. Nach regulärem Abschluß der OPEN-Anweisung existiert sie, ihr Status wird OLD.

SCRATCH — Die Datei ist temporär, d.h. ihre Existenz beginnt mit dem (regulären) Abschluß der OPEN-Anweisung und endet mit einer entsprechenden CLOSE-Anweisung (siehe 6.6), spätestens jedoch mit der Beendigung des Programms. Temporäre Dateien müssen unbenannt sein.

UNKNOWN — Der Status der Datei wird (anlagenabhängig) vom Rechner selbst bestimmt.

Die Werte OLD und NEW sind nur zulässig, wenn der Parameter FILE=fn angegeben ist; die Operation führt zu einem Fehler, wenn der tatsächliche Status der Datei dem angegebenen Wert nicht entspricht.

Fehlt der Parameter STATUS=sta ganz, wird STATUS='UNKNOWN' automatisch ergänzt.

6.3.3 Die Parameter ACCESS, FORM, RECL und BLANK

Die restlichen vier Parameter beschreiben die Eigenschaften einer Datei.

Der Parameter ACCESS=acc gibt die Art des Zugriffes an. acc ist ein Textausdruck, der, von zusätzlichen, nachfolgenden Leerzeichen abgesehen, linksbündig den Wert SEQUENTIAL oder DIRECT enthalten muß. Falls der Parameter nicht angegeben ist, wird der Parameter ACCESS='SEQUENTIAL' verwendet.
Bei einer Datei, die bereits existiert, muß die angegebene Art des Zugriffes für diese Datei zulässig sein.

Der Parameter FORM=fm gibt an, ob die Datei formatiert oder unformatiert ist. fm ist ein Textausdruck, der, von zusätzlichen, nachfolgenden Leerzeichen abgesehen, linksbündig den Wert FORMATTED oder UNFORMATTED enthalten muß. Falls der Parameter fehlt, wird für sequentielle Dateien der Parameter FORM='FORMATTED' und für direkte Dateien der Parameter FORM='UNFORMATTED' verwendet.

Nur für direkte Dateien darf und muß der Parameter RECL=rl angegeben werden. Der INTEGER-Ausdruck rl muß einen positiven Wert besitzen; er legt die Länge der Sätze der Datei fest, bei formatierten Sätzen gemessen in Zeichen und bei unformatierten Sätzen gemessen in Worten.

Nur für formatierte Dateien darf der Parameter BLANK=blnk angegeben werden. blnk ist ein Textausdruck, der, abgesehen von zusätzlichen, nachfolgenden Leerzeichen, den Wert NULL oder ZERO haben muß. Der Parameter beschreibt, ob beim formatgesteuerten Lesen eines numerischen Wertes Leerzeichen, die in der Zeichenfolge enthalten sind, ignoriert (BLANK='NULL') oder als Nullen (BLANK='ZERO') interpretiert werden sollen. Eine Zeichenfolge, die nur aus Leerzeichen besteht, ergibt in jedem Falle den Wert Null.
Falls der Parameter fehlt, wird BLANK='NULL' ergänzt.

6.4 Datenübertragung

6.4.1 READ und WRITE

Datenübertragungen aus einer Datei in den Hauptspeicher bewirkt die Anweisung

READ(stliste) [ealiste]

und Datenübertragungen aus dem Hauptspeicher in eine Datei die Anweisung

WRITE(stliste) [ealiste]

Als einziger Steuerparameter muß bei beiden Anweisungen

[UNIT=]u (siehe 6.2.2)

angegeben werden. Daneben können bei beiden Anweisungen angegeben werden:

[FMT=]f
REC=rn
IOSTAT=ios (siehe 6.2.3 bzw. 6.2.4)
ERR=s (siehe 6.2.3)

Nur bei READ-Anweisungen kann zusätzlich der Parameter

END=s (siehe 6.2.4)

angegeben werden.

Der Parameter [FMT=]f gibt an, daß es sich um eine formatierte Übertragung handelt. Die Angabe f kann ein Format bezeichnen oder ein Stern sein; im ersten Fall erfolgt die Übertragung formatgesteuert (siehe 7.1), im zweiten listengesteuert (siehe 7.2). Falls das Schlüsselwort FMT= weggelassen wird, muß f an zweiter Stelle in der Liste stehen, muß außerdem die Dateinummer u ohne das Schlüsselwort UNIT= an erster Stelle stehen. Falls der Parameter ganz fehlt, erfolgt eine unformatierte Übertragung.

Nur bei Zugriffen auf direkte Dateien darf und muß der Parameter REC=rn angegeben werden. Der Wert des INTEGER-Ausdrucks rn ergibt die Nummer des Satzes, der übertragen werden soll.

Beispiele:

(1)
```
WRITE(9) A
```

In die sequentielle Datei mit der Nummer 9 wird der Wert der Variablen A als ein Satz geschrieben.
Die gleiche Wirkung hat z.B. auch die Anweisungsfolge

```
I=3
WRITE(I+6) A
```

(2)
```
DIMENSION K(-5:5)
READ(END=17,FMT=*,UNIT=*) (K(I),I=-5,5,5)
```

Aus der Standard-Eingabedatei werden Werte für K(-5), K(0) und K(5) formatiert gelesen. Die Umwandlung erfolgt listengesteuert; falls bei der Operation der EOF-Satz gelesen wird, erfolgt ein Sprung zur Marke 17.
Kürzer kann man auch schreiben

```
DIMENSION K(-5:5)
READ(*,*,END=17) (K(I),I=-5,5,5)
```

Der erste Stern bezeichnet hier die Eingabe aus der Standard-Eingabedatei, der zweite die listengesteuerte Umwandlung.

(3)
```
DIMENSION A(100)
L=104
M=12
READ(M,REC=L,ERR=81) A
```

Aus der direkten Datei mit der Nummer 12 wird der Satz mit der Nummer 104 gelesen. Der Satz muß unformatiert sein und 100 REAL-Werte enthalten, die in die Elemente von A übertragen werden. Falls während des Lesens ein Fehler auftritt, erfolgt ein Sprung zur Marke 81.

6.4.2 Kurzformen

Speziell zum Lesen der Standard-Eingabedatei gibt es die Kurzform

```
READ f [,ealiste]
```

und zum Schreiben in die Standard-Ausgabedatei die Anweisung

```
PRINT f [,ealiste]
```

Sprachversionen für Rechner, bei denen die Möglichkeit zum Stanzen von Lochkarten besteht, kennen vielfach daneben zum Schreiben in eine Standard-Stanzdatei die Anweisung

```
PUNCH f [,ealiste]
```

Da diese Standard-Dateien immer formatiert sind, muß als Parameter f stets die Art der Formatierung angegeben werden, und zwar, wie bei der allgemeinen Form, entweder durch ein Format oder einen Stern. Weitere Steuerparameter können nicht angegeben werden.

Äquivalent sind so zum einen die beiden Anweisungen

```
READ(*,*) [ealiste]
READ * [,ealiste]
```

und zum anderen die beiden Anweisungen

```
WRITE(*,*) [ealiste]
PRINT * [,ealiste]
```

sofern die jeweiligen Ein-/Ausgabelisten übereinstimmen.

6.5 Positionierung sequentieller Dateien

Zur Positionierung sequentieller Dateien stehen die drei Anweisungen

```
BACKSPACE(stliste)
ENDFILE(stliste)
REWIND(stliste)
```

oder in Kurzform

```
BACKSPACE u
ENDFILE u
REWIND u
```

zur Verfügung.

Anzugeben ist immer der Parameter

[UNIT=]u (siehe 6.2.2)

Außerdem können bei Verwendung der allgemeinen Form die Parameter

IOSTAT=ios (siehe 6.2.3)
ERR=s (siehe 6.2.3)

angegeben werden.

Alle drei Anweisungen sind für die Standard-Ein-/Ausgabedateien nicht zulässig.

Die Anweisung BACKSPACE positioniert die Datei mit der Nummer u wieder auf den Anfang des Satzes, hinter dessen Ende sie momentan positioniert ist, d.h. die Datei wird um einen Satz "zurückgesetzt".
Die Datei mit der Nummer u muß existieren; falls sie auf den Anfang ihres ersten Satzes positioniert ist, hat die Anweisung keine Wirkung. Zulässig ist die Anweisung BACKSPACE nur dann, wenn der Satz, auf dessen Anfang die Datei positioniert werden soll, durch ein FORTRAN-Programm formatgesteuert geschrieben worden ist.

Die Anweisung REWIND positioniert die Datei mit der Nummer u auf den Anfang ihres ersten Satzes. Sie hat keine Wirkung, wenn die Datei dort bereits positioniert war.

Die Anweisung ENDFILE schreibt in die Datei mit der Nummer u einen EOF-Satz, und zwar an die Stelle, auf die die Datei gerade positioniert ist. Anschließend wird die Datei hinter den EOF-Satz positioniert.
Bevor aus der Datei oder in die Datei eine weitere Datenübertragung erfolgen kann, muß die Datei zunächst durch BACKSPACE- oder REWIND-Anweisungen wieder vor den EOF-Satz positioniert werden.

Beispiel: Zu einer (unbekannten) Anzahl von Meßwerten soll der Mittelwert berechnet werden. Außerdem sollen die Meßwerte und ihre Abweichungen vom Mittelwert aufgelistet werden. Die Meßwerte sollen jeweils als ein Satz der Standard-Eingabedatei vorliegen, die Ausgabe soll listengesteuert in die Standard-Ausgabedatei erfolgen. Zunächst müssen die Meßwerte gelesen und ihr Mittelwert gebildet werden. Da ihre Anzahl unbekannt ist, können sie nicht in einem

Feld gespeichert werden; da andererseits die Sätze der Standard-Eingabedatei nur einmal gelesen werden können, muß eine (temporäre) Datei als Hilfsspeicher verwendet werden.

```
      PROGRAM MWA
*     BERECHNUNG VON MITTELWERT UND ABWEICHUNGEN
      REAL MITTEL
*     ALS HILFSSPEICHER WIRD EINE TEMPORAERE,
*     SEQUENTIELLE, UNFORMATIERTE DATEI EROEFFNET
      OPEN(15,STATUS='SCRATCH',FORM='UNFORMATTED')
      MITTEL=0
      N=0
*     ZUNAECHST WERDEN DIE WERTE GELESEN, GEZAEHLT,
*     AUFSUMMIERT UND IN DIE DATEI 15 GESCHRIEBEN
    1 READ(*,*,END=2) WERT
      N=N+1
      MITTEL=MITTEL+WERT
      WRITE(15) WERT
      GOTO 1
*     DER MITTELWERT WIRD BERECHNET
    2 MITTEL=MITTEL/N
*     IN DIE DATEI 15 WIRD EIN EOF-SATZ GESCHRIEBEN
      ENDFILE 15
*     DIE DATEI 15 WIRD AUF IHREN ANFANG POSITIONIERT
      REWIND 15
*     DIE WERTE WERDEN ERNEUT GELESEN, DIE AUSGABE ERZEUGT
    3 READ(15,END=4) WERT
      ABW=ABS(MITTEL-WERT)
      PRINT *, 'WERT=',WERT,'  ABWEICHUNG=',ABW
      GOTO 3
    4 PRINT *, 'MITTELWERT=',MITTEL
      END
```

6.6 CLOSE

Die Anweisung

```
CLOSE(stliste)
```

löst die Zuordnung zwischen einer Datei und einer Dateinummer. Immer anzugeben ist der Steuerparameter

```
[UNIT=]u            (siehe 6.2.2)
```

Zusätzlich können die Steuerparameter

```
IOSTAT=ios          (siehe 6.2.3)
ERR=s               (siehe 6.2.3)
STATUS=sta
```

angegeben werden.

Mit dem Parameter STATUS=sta kann der Programmierer über die weitere Existenz der Datei verfügen. sta ist ein Textausdruck, der, abgesehen von zusätzlichen, nachfolgenden Leerzeichen, den Wert KEEP oder DELETE besitzen muß.
Der Wert KEEP ist nur für Dateien zulässig, die nicht mit dem Parameter STATUS='SCRATCH' eröffnet worden sind; er bewirkt, daß die Datei auch nach der CLOSE-Anweisung weiterhin existiert. Der Wert DELETE dagegen beendet die Existenz der Datei. Falls der Parameter STATUS=sta nicht angegeben ist, wird für SCRATCH-Dateien STATUS='DELETE', sonst STATUS='KEEP' verwendet.

Nachdem die Zuordnung gelöst wurde, können sowohl für die Datei, sofern sie noch existiert, als auch für die Dateinummer neue Zuordnungen vorgenommen oder auch die alte Zuordnung wiederhergestellt werden.

Beispiele:

(1)
```
OPEN(17,FILE='AB',STATUS='OLD')
   :
CLOSE(17)
OPEN(17,STATUS='SCRATCH',ACCESS='DIRECT',RECL=5)
```

Bis zur CLOSE-Anweisung bezeichnet die Dateinummer 17 die be-

reits existierende, sequentielle, formatierte Datei AB. Durch die CLOSE-Anweisung wird diese Zuordnung gelöst; die Datei AB existiert auch weiterhin.
Nach der zweiten OPEN-Anweisung bezeichnet die Dateinummer 17 eine temporäre, direkte, unformatierte, unbenannte Datei, deren Sätze jeweils 5 Worte lang sind.

(2)
```
OPEN(18,FILE='AB',STATUS='OLD')
   :
CLOSE(18,STATUS='DELETE')
OPEN(18,FILE='AB',STATUS='NEW')
```

Hier bezeichnet die Dateinummer 18 jeweils eine sequentielle, formatierte Datei mit dem Namen AB. Da die Existenz der ersten Datei mit der CLOSE-Anweisung endet, wird durch die zweite OPEN-Anweisung eine völlig andere Datei eröffnet. Insbesondere ist der Inhalt der ersten Datei nach der CLOSE-Anweisung wegen des Parameters STATUS='DELETE' verloren.

Am Ende eines Programms werden alle Zuordnungen, die durch das Programm vorgenommen und nicht bereits explizit wieder gelöst wurden, automatisch gelöst. Dieses erfolgt wie durch eine CLOSE-Anweisung ohne expliziten STATUS-Parameter, d.h. die Existenz temporärer Dateien wird beendet, andere Dateien existieren auch weiterhin.

6.7 INQUIRE

In vielen Fällen wird es beim Schreiben eines Programms unmöglich sein, vollständige Aussagen über Existenz und Eigenschaften der Dateien zu machen, die dem Programm zur Verfügung stehen müssen.
Soll z.B. ein Programm geschrieben werden, das auf eine bestimmte, anderweitig erstellte Datei zugreift, so ist es sinnvoll, sich zunächst zu vergewissern, daß
(1) die Datei tatsächlich existiert,
(2) ihr eine Dateinummer zugeordnet ist, und
(3) sie die erforderlichen Eigenschaften besitzt.
Die Möglichkeit, diese und andere Informationen zu beschaffen, bie-

tet die Anweisung

INQUIRE(stliste)

Anzugeben ist genau einer der beiden Steuerparameter

[UNIT=]u	(siehe 6.2.2)
FILE=fn	(siehe 6.3.1)

Je nachdem, welcher der beiden Parameter angegeben ist, spricht man von einer Abfrage nach Dateinummer (INQUIRE by unit) oder einer Abfrage nach Dateiname (INQUIRE by file).
Zusätzlich können die folgenden Parameter je nach Bedarf in beliebiger Kombination angegeben werden:

IOSTAT=ios	(siehe 6.2.3)
ERR=s	(siehe 6.2.3)
EXIST=ex	
OPENED=od	
NUMBER=num	
NAMED=nmd	
NAME=nm	
ACCESS=acc	
SEQUENTIAL=seq	
DIRECT=dir	
FORM=fm	
FORMATTED=fmt	
UNFORMATTED=unf	
RECL=rcl	
NEXTREC=nr	
BLANK=blnk	

Abgesehen von u bzw. fn und s sind bei den Parametern auf der rechten Seite jeweils Variable oder Feldelemente anzugeben, die während der Ausführung der Anweisung Werte erhalten.

6.7.1 Die Parameter EXIST, OPENED, NUMBER, NAMED und NAME

Bei dem Parameter EXIST=ex muß ex den Typ LOGICAL besitzen. ex erhält den Wert "wahr", wenn die Dateinummer u zulässig ist bzw. wenn eine Datei mit dem Namen fn existiert, und sonst den Wert "falsch".

Bei dem Parameter OPENED=od muß od den Typ LOGICAL besitzen. od erhält den Wert "wahr", wenn die Dateinummer u einer Datei bzw. wenn der Datei fn eine Dateinummer zugeordnet ist, und sonst den Wert "falsch".

Bei dem Parameter NUMBER=num muß num den Typ INTEGER besitzen. Geliefert wird als Resultat eine Dateinummer, falls od den Wert "wahr" besitzt, und sonst ein undefinierter Wert.

Bei dem Parameter NAMED=nmd muß nmd den Typ LOGICAL besitzen. nmd erhält den Wert "wahr", wenn entweder der Parameter FILE=fn angegeben oder die Datei mit der Nummer u benannt ist, und sonst den Wert "falsch".

Bei dem Parameter NAME=nm muß nm den Typ CHARACTER besitzen. Geliefert wird linksbündig in nm der Name der Datei, falls nmd den Wert "wahr" besitzt, und sonst ein undefinierter Wert.
Falls der Parameter FILE=fn angegeben ist, brauchen die Werte von fn und nm nicht übereinzustimmen. Die Norm schreibt nur vor, daß der Wert von nm ein Dateiname sein muß, der (anlagenabhängig) für die Angabe in einer OPEN-Anweisung zulässig ist. Dateien können u.U. mehrere, gleichwertige Namen haben; auch sind vielfach für Dateinamen Kurzformen zugelassen, die wie der Dateiname selbst verwendet werden können.

6.7.2 Die Parameter ACCESS, SEQUENTIAL und DIRECT

Bei den Parametern ACCESS=acc, SEQUENTIAL=seq und DIRECT=dir müssen acc, seq und dir den Typ CHARACTER besitzen. Alle möglichen Werte werden linksbündig gespeichert; eventuell vorhandene weitere Zeichenpositionen werden mit Leerzeichen gefüllt.

acc erhält den Wert SEQUENTIAL bzw. DIRECT zugewiesen, je nachdem, ob die Datei als sequentielle oder direkte Datei eröffnet wurde. Falls die Datei nicht eröffnet ist, erhält acc einen undefinierten Wert.

Mögliche Werte für seq und dir sind jeweils YES, NO und UNKNOWN. YES bzw. NO wird eingesetzt, wenn die Art des Zugriffes, die das Schlüsselwort des Parameters bezeichnet, für die Datei zulässig bzw. unzulässig ist, und UNKNOWN, wenn sich die Zulässigkeit bzw. Unzulässigkeit nicht feststellen läßt.

6.7.3 Die Parameter FORM, FORMATTED und UNFORMATTED

Bei den Parametern FORM=fm, FORMATTED=fmt und UNFORMATTED=unf müssen fm, fmt und unf den Typ CHARACTER besitzen. Alle möglichen Werte werden linksbündig gespeichert; eventuell vorhandene weitere Zeichenpositionen werden mit Leerzeichen gefüllt.

fm erhält den Wert FORMATTED bzw. UNFORMATTED zugewiesen, je nachdem, ob die Datei formatiert oder unformatiert ist. Falls die Datei nicht eröffnet ist, erhält fm einen undefinierten Wert.

Mögliche Werte für fmt und unf sind YES, NO und UNKNOWN. UNKNOWN wird eingesetzt, wenn sich nicht feststellen läßt, ob die Datei formatiert oder unformatiert ist; sonst erhält eine der Variablen fmt und unf den Wert YES, die andere den Wert NO.

6.7.4 Die Parameter RECL und NEXTREC

Bei den Parametern RECL=rcl und NEXTREC=nr erhalten rcl und nr immer dann undefinierte Werte, wenn die Datei nicht als direkte Datei eröffnet ist.

Falls die Datei als direkte Datei eröffnet ist, wird rcl die Länge der Sätze der Datei als INTEGER-Wert zugewiesen (siehe 6.3.3). nr erhält, ebenfalls als INTEGER-Wert, die Zahl n+1 zugewiesen, wenn n die Nummer des zuletzt gelesenen bzw. geschriebenen Satzes ist, bzw. den Wert 1, wenn seit der Eröffnung der Datei noch kein Satz

gelesen oder geschrieben wurde. Der Wert von nr ist undefiniert, wenn die Positionierung der Datei unbestimmt ist.

6.7.5 Der Parameter BLANK

Bei dem Parameter BLANK=blnk muß blnk den Typ CHARACTER besitzen.

Der Wert von blnk wird undefiniert, falls die Datei entweder überhaupt nicht oder nicht für formatierte Ein-/Ausgabe eröffnet ist. Falls die Datei für formatierte Ein-/Ausgabe eröffnet ist, wird in blnk linksbündig der Wert NULL oder ZERO gespeichert, je nachdem, wie Leerzeichen innerhalb von numerischen Feldern bei der Eingabe zu interpretieren sind (siehe 6.3.3).

7 Formatierung

7.1 Formatgesteuerte Umwandlung

Auf die Umwandlungen zwischen internen und externen Darstellungen, die bei formatierter Übertragung erforderlich werden, kann der Programmierer durch Formate direkt Einfluß nehmen. Ein Format hat die Form

([fliste])

mit einer Liste fliste von Formatbeschreibern (edit descriptor). Für die Angabe eines Formats bestehen mehrere Möglichkeiten:

(1) Das Format wird in einer Formatanweisung

f FORMAT([fliste])

mit einer Anweisungsnummer f definiert. Eine Formatanweisung muß immer mit einer Anweisungsnummer versehen sein.

Die Bezugnahme auf ein so erklärtes Format kann, nur in der gleichen Programmeinheit, in einer Ein-/Ausgabeanweisung im Steuerparameter [FMT=]f entweder durch die Angabe der Anweisungsnummer f selbst erfolgen, oder auch durch die Angabe einer INTEGER-Variablen i, der zuvor die Anweisungsnummer f durch eine Anweisung

ASSIGN f TO i

zugewiesen wurde.

(2) Im Steuerparameter [FMT=]f wird ein Textausdruck angegeben, dessen Wert ein Format ist. Man beachte: Textkonstanten, Textvariable, Elemente von Textfeldern und Teilketten können als (besonders einfache) Textausdrücke betrachtet werden; Konkatenationen sind nur zulässig, wenn kein Operand, abgesehen von benannten Konstanten, mit der Länge (*) vereinbart ist.

(3) Im Steuerparameter [FMT=]f wird der Name eines Textfeldes angegeben, dessen Elemente, wenn man sie insgesamt als fortlaufende Zeichenfolge betrachtet, als Wert ein Format enthalten. Die Größe des Feldes muß in der jeweiligen Programmeinheit explizit

vereinbart sein, d.h. in der Vereinbarung darf als obere Grenze für den Index der letzten Dimension des Feldes kein Stern angegeben sein.

Wesentlich ist, daß der korrekte Aufbau eines Formats erst dann überprüft wird, wenn das Format zur Steuerung einer Umwandlung verwendet wird. Es ist also ohne weiteres möglich, während der Ausführung eines Programms ein Format zunächst zu "berechnen" und danach zur Steuerung einer formatierten Übertragung darauf zuzugreifen.

7.1.1 Formatbeschreiber

Man unterscheidet zwei Arten von Formatbeschreibern:
Die Daten-Formatbeschreiber geben an, wie die externen Darstellungen der zu übertragenden Werte aussehen bzw. aussehen sollen. Zwischen ihnen und den Elementen der Ein-/Ausgabeliste wird eine (lineare) Zuordnung hergestellt.
Die Steuer-Formatbeschreiber geben zusätzliche Informationen unterschiedlicher Art für die Umwandlung. Ihnen werden keine Elemente der Ein-/Ausgabeliste zugeordnet.

Die Daten-Formatbeschreiber sind
(1) Iw und Iw.m für INTEGER-Werte,
(2) Fw.d, Ew.d, Ew.dEe, Dw.d, Gw.d und Gw.dEe für Gleitkomma-Werte,
(3) Lw für logische Werte, sowie
(4) Aw und A für Texte.
Hierin sind w, d, e und m jeweils vorzeichenlose INTEGER-Konstanten, müssen w und e stets positiv sein.
w gibt an, aus wie vielen Zeichen die externe Darstellung insgesamt besteht; d gibt für Gleitkomma-Werte an, wie viele der insgesamt w Zeichen Stellen hinter dem Dezimalpunkt sind.
Die Zeichenfolge, aus der die externe Darstellung eines Wertes besteht (wobei Leerzeichen mitzuzählen sind), wird auch als Eingabe-
SYSTEM WARNING - MAX PAGES feld bzw. als Ausgabefeld bezeichnet.

Unbedingt muß der Programmierer jeden Daten-Formatbeschreiber so wählen, daß er zu dem Element der Ein-/Ausgabeliste "paßt", dem er zugeordnet wird. Ungeeignete Beschreiber führen i.a. zu unvorher-

sagbaren Resultaten. Geeignet ist z.B. der Beschreiber Iw nur dann, wenn das Element der Ein-/Ausgabeliste, dem er zugeordnet wird, den Typ INTEGER besitzt.
Eine Besonderheit ist außerdem bei der Umwandlung komplexer Werte zu beachten: Dem Real- und Imaginärteil eines komplexen Elements in einer Ein-/Ausgabeliste wird jeweils ein eigener Beschreiber zugeordnet. Die beiden Beschreiber müssen jeweils für die Umwandlung von Gleitkomma-Werten geeignet sein; sie brauchen allerdings nicht identisch zu sein. Entsprechend werden für einen komplexen Wert zwei Eingabefelder bzw. zwei Ausgabefelder verwendet. Zwei Felder, die gemeinsam einen komplexen Wert darstellen, werden nicht besonders gekennzeichnet.

Die Steuer-Formatbeschreiber sind
(1) '$h_1h_2\ldots h_n$' und n$Hh_1h_2\ldots h_n$ zum Einfügen konstanter Zeichenfolgen,
(2) Tc, TLc, TRc und nX zur Positionierung innerhalb eines Satzes,
(3) / und : zur Steuerung der Übertragung,
(4) S, SP und SS zur Steuerung der Vorzeichen-Ausgabe,
(5) kP zur Skalierung numerischer Größen, sowie
(6) BN und BZ zur Steuerung, wie Leerzeichen in numerischen Eingabefeldern zu interpretieren sind.

Hierin sind n und c vorzeichenlose, positive INTEGER-Konstanten, k eine beliebige INTEGER-Konstante mit oder ohne Vorzeichen, sowie jedes h ein beliebiges Zeichen aus dem Zeichensatz des Rechners.

Je zwei Formatbeschreiber sind durch ein Komma voneinander zu trennen. Ausnahmen bilden die Steuer-Formatbeschreiber / und :, die anstelle eines Kommas stehen können, sowie der Steuer-Formatbeschreiber kP, dem kein Komma zu folgen braucht. Leerzeichen können, wie auch sonst in FORTRAN außerhalb von Textkonstanten, beliebig eingefügt werden.

7.1.2 Positionierung

Am Beginn einer Übertragung ist die Datei immer auf das erste Zeichen des zu übertragenden Satzes positioniert. Ein zu übertragendes

Datenfeld beginnt immer mit dem Zeichen, auf das die Datei gerade positioniert ist. Nach jeder Übertragung wird die Datei automatisch auf das Zeichen positioniert, das dem übertragenen Datenfeld folgt. Für die Praxis heißt das, daß aufeinanderfolgende Daten-Formatbeschreiber i.a. aufeinanderfolgende Datenfelder im Satz beschreiben.

Daneben gibt es aber auch die Möglichkeit, die Positionierung explizit zu verändern:

(1) Tc positioniert die Datei auf das c-te Zeichen des Satzes.
(2) TLc positioniert die Datei von ihrer momentanen Positionierung aus um c Zeichen nach links, jedoch nicht über den Anfang des Satzes hinaus.
(3) TRc bzw. nX positioniert die Datei von ihrer momentanen Positionierung aus um c bzw. n Zeichen nach rechts.

Eine explizite Positionierung bewirkt, für sich genommen, keine Datenübertragung.

Damit besteht beim Schreiben die Möglichkeit, in eine bereits übertragene Zeichenfolge "zurückzusetzen", dort ein (oder auch mehrere) Zeichen durch andere zu ersetzen, und anschließend wieder hinter das letzte bereits geschriebene Zeichen zu positionieren. Umgekehrt besteht beim Lesen die Möglichkeit, Teile eines Satzes mehrfach zu lesen.

Beim Schreiben in eine Datei mit fester Satzlänge (direkte Datei, interne Datei) und ebenso beim Lesen aus beliebigen Dateien darf eine explizite Positionierung auch über ein Satzende hinausführen; Daten dürfen dorthin bzw. von dort allerdings nicht übertragen werden.

Beim Schreiben in eine Datei mit variabler Satzlänge kann eine explizite Positionierung natürlich nicht über das Ende eines Satzes hinausführen, da die Länge des Satzes erst durch die Ausgabeoperation festgelegt wird. Die Länge eines Satzes ergibt sich jedoch ausschließlich aus den vorgenommenen Datenübertragungen und wird durch explizite Positionierungen nicht beeinflußt.

Beim Schreiben können explizite Positionierungen indirekt auch die Übertragung von Daten, nämlich von Leerzeichen bewirken: Bevor die

Übertragung in einen Satz abgeschlossen wird, wird in jede seiner Zeichenpositionen, in die bislang noch kein Zeichen übertragen worden ist, ein Leerzeichen übertragen. Hiervon können Zeichenpositionen betroffen sein, die bei einer expliziten Positionierung "übersprungen" worden sind, sowie, bei Dateien mit fester Satzlänge, Zeichenpositionen hinter dem letzten übertragenen Zeichen.

7.1.3 Eingabe numerischer Werte

Grundsätzlich gilt für die formatgesteuerte Eingabe numerischer Werte:

(1) Führende Leerzeichen sind nicht signifikant.

(2) Sonstige Leerzeichen werden entsprechend dem BLANK-Parameter der OPEN-Anweisung ignoriert oder als Nullen interpretiert. Der BLANK-Parameter kann durch einen Steuer-Formatbeschreiber BN (entspricht BLANK='NULL') bzw. BZ (entspricht BLANK='ZERO') geändert werden. Änderungen durch BN bzw. BZ gelten immer nur für die jeweilige Eingabeoperation; nach Beendigung der Eingabeoperation wird wieder der BLANK-Parameter der OPEN-Anweisung wirksam.

(3) Ein Eingabefeld, das nur Leerzeichen enthält, repräsentiert den Wert Null.

Zu den Beschreibern im einzelnen.

Die Beschreiber Iw und Iw.m sind bei der Eingabe gleichwertig.
Beispiel: Wird der Satz

1␣1␣1␣1␣1

gelesen durch

```
READ(14,'(BN,I3,I3,BZ,I3)') J1,J2,J3
```

so erfolgen die Wertzuweisungen J1=11, J2=1 und J3=101.

Ebenfalls gleichwertig sind bei der Eingabe die Beschreiber Fw.d, Ew.d, Ew.dEe, Dw.d, Gw.d und Gw.dEe.
Gleitkomma-Werte können im Eingabefeld dargestellt werden durch ei-

ne (möglicherweise leere) Ziffernfolge mit oder ohne Vorzeichen mit oder ohne Dezimalpunkt sowie ggf. mit einem nachfolgenden Exponententeil. Dieser Exponententeil kann entweder aus einer INTEGER-Konstanten mit Vorzeichen bestehen oder aus einem der Kennbuchstaben E oder D und einer nachfolgenden INTEGER-Konstanten mit oder ohne Vorzeichen.
Während eine eventuelle Angabe Ee bei der Eingabe grundsätzlich keine Bedeutung hat, wird die Angabe d dann und nur dann ignoriert, wenn die Ziffernfolge, die den Eingabewert repräsentiert, einen Dezimalpunkt enthält. Falls die Ziffernfolge keinen Dezimalpunkt enthält, werden ihre letzten d Stellen vor Beginn des Exponententeiles als Stellen hinter dem Dezimalpunkt betrachtet. Leerzeichen werden hierbei nur mitgezählt, wenn sie als Nullen zu interpretieren sind.

Beispiele:

(1) Mit dem Eingabesatz

␣12␣345␣67

liefert die Anweisung

```
READ '(F3.1,F4.2,F3.2)', A1,A2,A3
```

die Wertzuweisungen A1=1.2, A2=3.45 und A3=6.7. Mit dem Eingabesatz

.12.345.67

liefert dieselbe Eingabeanweisung die Wertzuweisungen A1=.12, A2=.345 und A3=.67.

(2) Mit dem Eingabesatz

1234E12345

liefert die Anweisung

```
READ '(E6.1,F4.2)', A1,A2
```

die Wertzuweisungen A1=123.4E+1 und A2=23.45, während die Anweisung

```
READ '(E7.1,E3.2)', A1,A2
```

mit demselben Eingabesatz die Wertzuweisungen A1=123.4E12 und

A2=3.45 liefert.

(3) Mit dem Eingabesatz

.123E12345

liefert die Anweisung

READ '(E6.1,F4.2)', A1,A2

die Wertzuweisungen A1=.123E1 und A2=23.45.

7.1.4 Ausgabe numerischer Werte

Grundsätzlich erfolgt die formatgesteuerte Ausgabe numerischer Werte nach folgenden Regeln:

(1) Der Wert wird rechtsbündig in ein Feld übertragen, dessen Größe durch w bestimmt wird. Führende, nicht signifikante Nullen werden durch Leerzeichen ersetzt.
Ausnahmen gibt es bei Iw.m, Gw.d und Gw.dEe.

(2) Bei Gleitkomma-Werten wird die letzte auszugebende Stelle gerundet.

(3) Bei halblogarithmisch darzustellenden Gleitkomma-Werten besteht der Exponententeil immer aus einer INTEGER-Konstanten mit Vorzeichen. Teilweise wird dem Vorzeichen noch ein Kennbuchstabe vorangestellt.

(4) Bei der halblogarithmischen Ausgabe von Gleitkomma-Werten ohne Skalenfaktor (siehe 7.1.10) wird der Exponent immer so gewählt, daß für die Mantisse m gilt

$0.1 \leq |m| < 1$

Die Norm verlangt nicht, daß in der Mantisse links neben den Dezimalpunkt eine Null gesetzt wird, läßt es aber zu.

(5) Vorzeichen werden unmittelbar links neben die werthöchste Ziffer bzw. den Dezimalpunkt gesetzt.
Nur negative Werte erhalten immer ein Vorzeichen. Für positive Werte enthält die Norm keine Vorschrift; der (anlagenabhängige) Standard kann jedoch durch Steuer-Formatbeschreiber aufgehoben

werden: SP bewirkt, daß alle Werte ein Vorzeichen erhalten, und SS, daß nur negative Werte ein Vorzeichen erhalten. Wirksam bleiben derartige Änderungen der Standardform höchstens bis zum Ende der Ausgabeoperation. Vorzeitig können sie durch eine neuerliche Änderung aufgehoben werden; vorzeitig kann aber auch durch den Steuer-Formatbeschreiber S der ursprüngliche Zustand wiederhergestellt werden.

(6) Jedes Ausgabefeld muß so groß gewählt werden, daß die gesamte auszugebende Zeichenfolge hineinpaßt, also zusätzlich zu der eigentlichen Ziffernfolge ggf. Vorzeichen, Dezimalpunkt und Exponententeil. Falls ein Ausgabefeld zu klein gewählt ist, wird es in voller Länge mit Sternen gefüllt.

Zu den Beschreibern im einzelnen.

Während bei dem Beschreiber Iw führende Nullen in den ersten w-1 Stellen durch Leerzeichen ersetzt werden, geschieht dieses bei dem Beschreiber Iw.m ($w \geq m$) nur in den ersten w-m Stellen, so daß u.U. führende Nullen erhalten bleiben.
Wird der Wert Null ausgegeben, so erscheint bei Iw rechtsbündig im Ausgabefeld eine Null; bei Iw.m können mehrere Nullen erscheinen ($m>0$) oder auch das Ausgabefeld ganz leer sein ($m=0$). In allen Fällen kann das ganz rechts stehende Leerzeichen durch ein positives Vorzeichen ersetzt sein.

Gleitkomma-Darstellungen ohne Exponententeil erzeugt der Beschreiber Fw.d.

Gleitkomma-Darstellungen mit Exponententeil erzeugen die Beschreiber Ew.d, Ew.dEe und Dw.d. Für die Darstellung eines Exponententeiles exp läßt die Norm verschiedene Möglichkeiten zu, teilweise abhängig von seinem Wert:

Beschreiber	Wert von exp	Darstellung
Ew.d	$\|exp\| \leq 99$	E±zz ±0zz
	$99 < \|exp\| \leq 999$	±zzz
Ew.dEe	beliebig	E±$z_1 \ldots z_e$
Dw.d	$\|exp\| \leq 99$	D±zz E±zz ±0zz
	$99 < \|exp\| \leq 999$	±zzz

Hierin steht jedes z für eine Ziffer. Man beachte: Die Beschreiber Ew.d und Dw.d sind für Werte mit vierstelligen Exponenten (sofern diese überhaupt vorkommen können) nicht zulässig.

In den Fällen, in denen mehrere zulässige Darstellungen angegeben sind, kann die Darstellung von Anlage zu Anlage verschieden sein. Der Programmierer kann zwischen den verschiedenen Darstellungen nicht wählen.

Die Beschreiber Gw.d und Gw.dEe erzeugen, abhängig vom Ausgabewert R, Gleitkomma-Darstellungen mit oder ohne Exponententeil. Falls |R|<0.1 oder |R|≥10**d gilt, erzeugt Gw.d bzw. Gw.dEe dieselbe externe Darstellung wie Ew.d bzw. Ew.dEe, also eine halblogarithmische Darstellung mit Exponententeil. Sonst wird eine Darstellung ohne Exponententeil mit genau d signifikanten Stellen erzeugt, in die der Dezimalpunkt an der richtigen Stelle eingefügt ist. Bei Darstellungen ohne Exponententeil erfolgt die Ausgabe nicht rechtsbündig; vielmehr werden im Fall Gw.d die letzten 4 und im Fall Gw.dEe die letzten e+2 Stellen des Ausgabefeldes mit Leerzeichen gefüllt.

Beispiele:

Beschreiber	interner Wert	externe Darstellung
I3	-5	␣-5
	+5	␣␣5
		␣+5
	137	137
	-137	***
SS,I3	+5	␣␣5
SP,I3	+5	␣+5
SP,I3.0	0	␣␣+
F7.2	-12.318	␣-12.32
E10.2	-12.318	␣␣-.12E+02
		␣␣-.12+002
D10.2	-12.318	␣␣-.12D+02
		␣␣-.12+002
E10.2E1	-12.318	␣␣␣-.12E+2
G10.4	+17.3	␣17.30␣␣␣
	+.0173	␣.1730E-01
	-17300.	-.1730E+05
G10.4E1	+17.3	␣␣17.30␣␣␣
	+.0173	␣␣.1730E-1
	-17300.	␣-.1730E+5
	+1730000000.	**********

7.1.5 Ein-/Ausgabe logischer Werte

Zur Ein-/Ausgabe logischer Werte dient der Beschreiber Lw.

Das Eingabefeld muß entweder den Buchstaben T, der den Wert "wahr" ergibt, oder den Buchstaben F, der den Wert "falsch" ergibt, enthalten. Unmittelbar vor dem Buchstaben darf ein Punkt, davor eine beliebig lange Folge von Leerzeichen stehen. Hinter dem Buchstaben T bzw. F dürfen beliebige Zeichen folgen. Zulässig ist also insbesondere, daß ein Eingabefeld nur einen Buchstaben T bzw. F oder eine FORTRAN-Konstante .TRUE. bzw. .FALSE. enthält.

Bei der Ausgabe werden rechtsbündig in das Ausgabefeld der Buchsta-

be T bzw. F und davor Leerzeichen geschrieben.

Beispiel: Mit dem Eingabesatz

```
.TRUE.FALSCHbbbb.Fb.TAGE
```

schreibt die Anweisungsfolge

```
LOGICAL L(4)
READ '(L6,L6,L6,L6)', L
WRITE(19,'(L4,L3,L2,L1)') L
```

in die Datei mit der Nummer 19 den Ausgabesatz

```
bbbTbbFbFT
```

7.1.6 Ein-/Ausgabe von Texten

Zur Ein-/Ausgabe von Texten dienen die Daten-Formatbeschreiber Aw und A.

Bei der Ein-/Ausgabe mit Aw hängt die Übertragung sowohl von der Größe w des Ein-/Ausgabefeldes als auch von der Länge l des Elements der Ein-/Ausgabeliste ab:

(1) Eingabe:

- w<l Die w Zeichen des Eingabefeldes werden linksbündig in das Eingabeelement übertragen. In die restlichen l-w Zeichenpositionen des Eingabeelements werden Leerzeichen übertragen.
- w=l Die w Zeichen des Eingabefeldes ergeben den Wert des Eingabeelements.
- w>l Die linken w-l Zeichen des Eingabefeldes werden nicht übertragen; die restlichen l Zeichen ergeben den Wert des Eingabeelements.

(2) Ausgabe:

- w<l Die linken w Zeichen des Ausgabeelements ergeben den Inhalt des Ausgabefeldes; die restlichen l-w Zeichen des Ausgabeelements werden nicht übertragen.

w=1 Die 1 Zeichen des Ausgabeelements ergeben den Inhalt des Ausgabefeldes.

w>1 In das Ausgabefeld werden w-1 Leerzeichen und daran anschließend die 1 Zeichen des Ausgabeelements übertragen.

Wesentlich bequemer ist in vielen Fällen die Verwendung des Beschreibers A. Hier bestimmt der Rechner zunächst die Länge l des Ein-/Ausgabeelements und verwendet sie anschließend als Länge des Ein-/Ausgabefeldes. Für ein Ein-/Ausgabeelement mit der Länge 1 hat der Beschreiber A also die gleiche Wirkung wie der Beschreiber A1.

Beispiel: Das Pascalsche Dreieck soll in Pyramidenform gedruckt werden. Das Programm

```
      PROGRAM PASCAL
      CHARACTER LEER*28
      INTEGER ZEILE(0:9)
      LEER=' '
      DO 2 I=0,9,1
      ZEILE(I)=1
      DO 1 J=I-1,1,-1
    1 ZEILE(J)=ZEILE(J)+ZEILE(J-1)
    2 PRINT '(A,10I6)', LEER(1:(9-I)*3+1),
     *                  (ZEILE(J),J=0,I,1)
      END
```

erzeugt die verlangte Ausgabe in der Form

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
```

Nur bei der Ausgabe, nicht bei der Eingabe, besteht außerdem die Möglichkeit, Texte als Steuer-Formatbeschreiber im Format anzugeben, und zwar wahlweise in der Form einer Textkonstanten

$'h_1h_2\cdots h_n'$

oder in der Form

$nHh_1h_2\cdots h_n$

worin jedes h ein Zeichen aus dem Zeichenvorrat des Rechners bezeichnet. Der Wert einer solchen Textkonstanten wird wie ein Ausgabewert in ein Feld des Ausgabesatzes übertragen.

Man beachte: Für einen Apostroph als Teil der Zeichenfolge gelten bei Verwendung der ersten Form die gleichen Regeln wie sonst auch für Textkonstanten; bei Verwendung der zweiten Form kann ein Apostroph wie ein beliebiges anderes Zeichen angegeben werden. Falls allerdings ein Format selbst als Textkonstante angegeben wird, sind alle Apostrophe im Format zu verdoppeln; z.B. kann das Format (6H F'(X)) als Textkonstante dargestellt werden durch '(6H F''(X))' oder durch '('' F''''(X)'')'. Da die zusätzlichen Apostrophe ausschließlich zur Kennzeichnung dienen, verändern sie die Länge n eines Beschreibers nH natürlich nicht.

Beispiel: Komplexe Werte erscheinen bei formatgesteuerter Ausgabe gemeinhin als zwei, voneinander unabhängige Gleitkomma-Werte. Soll ein komplexer Wert als FORTRAN-Konstante geschrieben werden, also eingeschlossen in Klammern und mit einem Komma zwischen Real- und Imaginärteil, so kann man etwa programmieren

```
COMPLEX C
C=(1.5E-4,-2.1E+3)
WRITE(17, '(2H (,E9.3E2,1H,,E9.3E2,1H))' ) C
```

Der Satz, der hier in die Datei mit der Nummer 17 übertragen wird, besteht aus der Zeichenfolge

b(b.150E-03,-.210E+04)

7.1.7 Drucken

Eine Besonderheit ist bei der Ausgabe auf einen Zeilendrucker zu beachten: Das erste Zeichen jedes Ausgabesatzes steuert die Vorschübe des Druckers und erscheint selbst nicht in der Liste; die weiteren Zeichen eines Satzes, sofern vorhanden, bilden eine Zeile.

Zur Verfügung stehen vier Vorschubsteuerzeichen:

b	einfacher Zeilenvorschub
0	doppelter Zeilenvorschub
+	kein Vorschub ("Überdrucken")
1	Vorschub in die erste Zeile einer neuen Seite

Die Norm läßt offen, was passiert, wenn ein Satz auf einen Zeilendrucker ausgegeben wird, der in seiner ersten Zeichenposition keines dieser vier Zeichen enthält.

Grundsätzlich erfolgt immer zuerst der Vorschub, danach die Ausgabe der restlichen Zeichen des Satzes in die momentane Zeile. Besteht ein Satz nur aus einem Vorschubsteuerzeichen, so wird der Vorschub normal ausgeführt, danach eine Leerzeile ausgegeben; enthält ein Ausgabesatz überhaupt kein Zeichen, so wird eine Leerzeile mit einfachem Zeilenvorschub ausgegeben.

Prinzipiell bleibt es dem Programmierer selbst überlassen, wie er in einen zu druckenden Satz als erstes Zeichen ein Vorschubsteuerzeichen hineinschreibt. Es empfiehlt sich jedoch, im Normalfall das Vorschubsteuerzeichen in Form einer Textkonstanten als Steuer-Formatbeschreiber anzugeben. So bewirkt zum Beispiel die Anweisung

```
PRINT '(1H1,I1)', 1
```

daß in die erste Zeichenposition der ersten Zeile einer neuen Seite eine Eins gedruckt wird. Die gleiche Wirkung hat zwar auch die Anweisung

```
PRINT '(I2)', 11
```

jedoch ist vor derartiger "Trickprogrammierung" nachdrücklich zu warnen.

Bei verschiedenen Rechenanlagen kann die Vorschubsteuerung sehr un-

terschiedlich realisiert sein. Bei manchen Anlagen erfolgt eine Vorschubsteuerung bei Zeilendruckern durch das erste Zeichen jedes Ausgabesatzes nur dann, wenn die jeweilige Datei speziell als Druckdatei erklärt ist, z.B. durch einen zusätzlichen, anlagenspezifischen Steuerparameter der OPEN-Anweisung. Bei anderen Anlagen erfolgt eine Vorschubsteuerung durch das erste Zeichen jedes Ausgabesatzes bei jeder Übertragung einer formatierten Datei auf einen Zeilendrucker, kann also jede formatierte Datei als Druckdatei angesehen werden.

Erhebliche Unterschiede bestehen auch, wenn eine Druckdatei statt auf einen Zeilendrucker auf ein Bildschirmgerät ausgegeben oder durch ein FORTRAN-Programm wieder gelesen wird.
Bei der Ausgabe auf ein Bildschirmgerät werden die Vorschubsteuerzeichen bei manchen Anlagen als solche interpretiert, bei anderen Anlagen wie die restlichen Zeichen ausgegeben und bei wieder anderen Anlagen völlig ignoriert. Falls die Vorschubsteuerzeichen als solche interpretiert werden, können andere Vorschübe als bei einem Zeilendrucker resultieren; z.B. lassen Bildschirmgeräte i.a. ein "Überdrucken" nicht zu.
Beim Lesen einer Druckdatei durch ein Programm kann ein Vorschubsteuerzeichen entweder als erstes Datenzeichen des jeweiligen Satzes behandelt oder auch ignoriert werden. Wurde z.B. in eine Druckdatei mit der Nummer 17 die Zeichenfolge 12$ als ein Satz geschrieben, so kann die Anweisung

```
READ(17, '(BN,I2)' ) J
```

mit diesem Satz als Eingabe bei einer Anlage die Wertzuweisung J=2, bei einer anderen die Wertzuweisung J=12 bewirken.

Den jeweiligen Benutzerhandbüchern muß entnommen werden, wie die spezielle Rechenanlage Druckdateien behandelt.

7.1.8 Wiederholungsfaktoren

Falls in einem Format identische Daten-Formatbeschreiber unmittelbar aufeinanderfolgen oder sich Folgen von (beliebigen) Formatbeschreibern wiederholen, brauchen diese nicht einzeln explizit angegeben zu werden, sondern können durch Wiederholungsfaktoren erzeugt werden. Wiederholungsfaktoren sind vorzeichenlose INTEGER-Konstanten ungleich Null.

Daten-Formatbeschreibern können Wiederholungsfaktoren unmittelbar vorangestellt werden. So hat 3I4 die gleiche Bedeutung wie die Folge I4,I4,I4.

Eine Gruppe von beliebigen Formatbeschreibern kann wiederholt werden, indem sie in Klammern gesetzt und vor die linke Klammer ein Wiederholungsfaktor geschrieben wird. So hat 2(5X,F9.2,'**',I2) dieselbe Bedeutung wie die Folge 5X,F9.2,'**',I2,5X,F9.2,'**',I2.

Klammern dürfen geschachtelt werden. Auch wenn die Norm keine maximale Tiefe für die Schachtelung festlegt, existieren in der Praxis teilweise Beschränkungen. So läßt CDC-FORTRAN maximal die Schachtelungstiefe 9 und FORTRAN (ASCII) von UNIVAC maximal die Schachtelungstiefe 5 zu; dabei sind die Klammern mitzuzählen, die das Format insgesamt einschließen.

Klammern dürfen aber auch gesetzt werden, ohne daß ihnen ein Wiederholungsfaktor vorangestellt wird. Ein fehlender Wiederholungsfaktor hat, wie bei den Daten-Formatbeschreibern, die gleiche Wirkung wie der Wiederholungsfaktor 1. Die Bedeutung von Klammern ohne expliziten Wiederholungfaktor wird im folgenden Abschnitt beschrieben.

7.1.9 Zusammenhang zwischen Ein-/Ausgabeliste und Format

Die Abarbeitung einer formatgesteuerten Datenübertragung wird teils durch die Ein-/Ausgabeliste und teils durch das Format bestimmt.

Falls eine Ein-/Ausgabeliste mindestens ein Element enthält, muß das zugeordnete Format mindestens einen Daten-Formatbeschreiber

enthalten. Man beachte: Ein leeres Format darf nur einer leeren Ein-/Ausgabeliste zugeordnet werden. Es bewirkt die Übertragung eines leeren Satzes; bei der Eingabe bedeutet das, daß ein Eingabesatz übersprungen wird, und beim Drucken, daß eine Leerzeile ausgegeben wird.

Grundsätzlich wird eine Datei am Beginn einer formatierten Übertragung auf den Anfang eines (neuen) Satzes positioniert. Bei formatgesteuerter Übertragung beginnt anschließend die Interpretation des Formats.

Die Interpretation eines Formats und gleichzeitig damit die Übertragungsoperation wird beendet, sobald festgestellt wird, daß die Ein-/Ausgabeliste vollständig abgearbeitet ist. Überprüft wird die Ein-/Ausgabeliste in drei Fällen, nämlich

(1) wenn bei der Interpretation des Formats ein Daten-Formatbeschreiber erkannt wird,
(2) wenn die abschließende Klammer des Formats erreicht wird, und
(3) wenn im Format ein Doppelpunkt gefunden wird.

Die anderen Steuer-Formatbeschreiber werden dagegen stets abgearbeitet, ohne daß zunächst geprüft wird, ob die Ein-/Ausgabeliste abgearbeitet ist.

Ist etwa das Format

```
17 FORMAT(' I=',I3,' J=',I3)
```

vorgegeben, so erzeugt die Anweisung

```
PRINT 17, 5,9
```

in der Standard-Ausgabedatei den Satz

```
␣I=␣␣5␣J=␣␣9
```

und die Anweisung

```
PRINT 17, 5
```

ebenfalls in der Standard-Ausgabedatei den Satz

```
␣I=␣␣5␣J=
```

Ist dagegen das Format

```
17 FORMAT(' I=',I3:' J=',I3)
```

vorgegeben, so erzeugt die Anweisung

```
PRINT 17, 5,9
```

denselben Satz wie oben, die Anweisung

```
PRINT 17, 5
```

dagegen den Satz

```
 I=  5
```

Während bei jedem Zugriff auf eine direkte Datei und bei jedem Zugriff auf eine unformatierte sequentielle Datei mit jeder Übertragungsoperation genau ein Satz übertragen wird, können mit einem Zugriff auf eine formatierte, sequentielle Datei auch mehrere Sätze übertragen werden.

Wenn im Format ein Schrägstrich (slash) gefunden wird, wird ein Satz abgeschlossen und ein neuer begonnen. So erzeugt die Anweisung

```
PRINT '(3H1I=,I3/3HOJ=,I3)', 5,9
```

die beiden Sätze

```
1I=  5
OJ=  9
```

Falls diese Datei gedruckt wird, ergeben die beiden Sätze am Kopf einer neuen Seite die drei Zeilen

```
I=  5
 
J=  9
```

Ebenfalls wird ein Satz beendet, wenn die abschließende Klammer des Formats erreicht wird. Sofern die Ein-/Ausgabeliste dann noch nicht abgearbeitet ist, wird ein neuer Satz begonnen und das Format entweder teilweise oder auch in voller Länge neu interpretiert.
Ganz von vorne wird es interpretiert, wenn es keine in Klammern gesetzte Folge von Beschreibern enthält. Sonst beginnt die Interpretation mit dem Wiederholungsfaktor des Klammernpaares, das durch die vorletzte Klammer im Format abgeschlossen wird. Dabei ist das Klammernpaar mitzuzählen, das das Format insgesamt einschließt.

Beispiel: Zu drucken ist eine Funktionstabelle für sin(x) für $x \in [0,1]$ mit der Schrittweite 0.1. Das Programm

```
      PROGRAM SINTAB
    1 FORMAT('1Tabelle für sin(x)'/
     *        '0   x    sin(x)'//
     *        (F5.1,F10.6))
      PRINT 1, (X,SIN(X),X=0.,1.,.1)
      END
```

liefert diese Tabelle in der Form

```
Tabelle für sin(x)
ƀ
   x    sin(x)
ƀ
  .0   .000000
  .1   .099833
  .2   .198669
  .3   .295520
  .4   .389418
  .5   .479426
  .6   .564642
  .7   .644218
  .8   .717356
  .9   .783327
 1.0   .841471
```

beginnend in der ersten Zeile einer neuen Seite.

Man beachte:

(1) Das erste Zeichen jedes Satzes wird beim Drucken auch dann zur Vorschubsteuerung verwendet, wenn mehrere Sätze mit einer Anweisung gedruckt werden.

(2) Ein Schrägstrich, der als erster oder letzter Beschreiber in einem Format interpretiert wird, bewirkt immer, daß ein leerer Satz übertragen wird.

(3) Die in diesem Abschnitt genannten Regeln gelten sowohl für die Ausgabe als auch für die Eingabe, auch wenn die Beispiele nur

die Ausgabe behandeln.

7.1.10 Skalenfaktoren

Durch Skalenfaktoren können die meisten Umwandlungen von Gleitkomma-Werten beeinflußt werden.

Bei Beginn einer formatgesteuerten Übertragung ist ein Skalenfaktor 0P wirksam; dieser kann durch die explizite Angabe eines Skalenfaktors kP überschrieben werden. Ein explizit gesetzter Skalenfaktor bleibt wirksam, bis er durch einen weiteren, explizit angegebenen Skalenfaktor überschrieben wird, höchstens jedoch bis zum Ende der jeweiligen Übertragungsoperation.

Bei der Eingabe bewirkt ein Skalenfaktor kP, daß jeder ohne Exponententeil dargestellte Gleitkomma-Wert mit 10**(-k) multipliziert wird; halblogarithmisch dargestellte Eingabewerte bleiben unverändert.

Beispiele:

Externe Darstellung	Format	Interner Wert
4.32␣␣	0PE6.2	4.32
4.32␣␣	1PE6.2	.432
4.32␣␣	-2PE6.2	432.
4.32E0	-2PE6.2	4.32

Umgekehrt bewirkt ein Skalenfaktor kP bei der Ausgabe mit Fw.d, daß der Ausgabewert mit 10**k multipliziert wird.

Beispiele:

Interner Wert	Format	Externe Darstellung
4.32	0PF7.4	␣4.3200
4.32	1PF7.4	43.2000
4.32	-1PF7.4	␣␣.4320

Bei der Ausgabe mit den Beschreibern Gw.d und Gw.dEe bleiben Skalenfaktoren wirkungslos, sofern Darstellungen ohne Exponententeil erzeugt werden.

Bei der Ausgabe von halblogarithmisch darzustellenden Gleitkomma-

Werten bewirkt ein Skalenfaktor kP bei allen dafür vorgesehenen Beschreibern, daß die Mantisse mit 10**k multipliziert, der Exponent um k verringert wird.

Beispiele:

Interner Wert	Format	Externe Darstellung
4.32	E11.4	␣␣.4320E+01
4.32	1PE11.4	␣4.3200E+00
4.32	-1PE11.4	␣␣.0432E+02

Man beachte: Bei Gleitkomma-Werten, die extern ohne Exponententeil dargestellt werden, kann ein Skalenfaktor bewirken, daß interne und externe Darstellung verschiedene Werte repräsentieren. Bei der Ausgabe halblogarithmisch zu schreibender Gleitkomma-Werte ändert ein Skalenfaktor dagegen nur die Stellung des Dezimalpunktes innerhalb der Mantisse, nicht jedoch den Wert der Zahl.

7.2 Listengesteuerte Umwandlung

Listengesteuerte Umwandlung wird im Steuerparameter [FMT=]f durch einen Stern beschrieben.
Man beachte: Bei manchen Rechnern bewirkt auch die Angabe eines leeren Formats listengesteuerte Umwandlung. Dieses steht jedoch im Widerspruch zu den Vorschriften der Norm (siehe 7.1.9).

Listengesteuerte Umwandlung ist nur für externe, sequentielle Dateien zulässig; Sätze, die listengesteuert geschrieben wurden, können mit BACKSPACE nicht übersprungen werden.

7.2.1 Listengesteuerte Eingabe

Die Werte in einem Eingabesatz müssen im wesentlichen wie für eine formatgesteuerte Eingabe dargestellt sein; nur muß zwischen je zwei Werten ein Trennzeichen stehen. Trennzeichen sind
(1) das Komma, dem ein Leerzeichen voranstehen oder folgen kann,
(2) der Schrägstrich (slash), dem ein Leerzeichen voranstehen oder folgen kann, sowie

(3) das Leerzeichen selbst, wenn es zwischen zwei Werten steht, die nicht zusätzlich durch ein anderes Trennzeichen getrennt werden.

Das Ende eines Eingabesatzes wird, außer wenn es in einer Textkonstanten steht, wie ein Leerzeichen behandelt; ebenso werden beliebig lange Folgen von Leerzeichen, die nicht Bestandteile von Textkonstanten sind, wie ein Leerzeichen behandelt.
Man beachte: Leerzeichen, die bei formatgesteuerter Eingabe wahlweise ignoriert oder als Nullen interpretiert werden, beenden bei listengesteuerter Eingabe immer ein Eingabefeld.

Anhand der Trennzeichen bestimmt der Rechner Eingabefelder und ordnet diese den Elementen der Eingabeliste zu. Der Programmierer hat darauf zu achten, daß jeder Eingabewert eine Darstellung besitzt, die für den Typ des Eingabeelements, dem er zugeordnet wird, zulässig ist.

Jede listengesteuerte Eingabeanweisung liest mindestens einen Satz. Weitere Sätze werden bei Bedarf gelesen, bis jedem Eingabeelement ein Eingabefeld zugeordnet ist.

Unterschiede zur formatgesteuerten Eingabe bestehen in den folgenden Punkten:

(1) Bei nicht ganzzahligen Gleitkomma-Werten muß der Dezimalpunkt immer explizit angegeben werden, bei ganzzahligen Gleitkomma-Werten kann er fehlen.

(2) Komplexe Werte müssen in Klammern gesetzt, ihr Real- und Imaginärteil durch ein Komma voneinander getrennt werden. Sowohl den Klammern als auch dem Komma dürfen Leerzeichen vorangestellt werden und folgen.

(3) Bei logischen Konstanten darf die Zeichenfolge, die dem Buchstaben T bzw. F folgen kann, kein Trennzeichen enthalten, also kein Komma, kein Leerzeichen und keinen Schrägstrich.

(4) Texte müssen jeweils durch einen kennzeichnenden Apostroph eingeleitet und abgeschlossen werden. Falls die Textkonstante im Eingabesatz und das Eingabeelement unterschiedliche Längen besitzen, erfolgt die Übertragung wie bei einer Wertzuweisung

(siehe 5.2.2).
Ein Apostroph innerhalb der Zeichenfolge ist durch zwei aufeinanderfolgende Apostrophe darzustellen. Zwischen den beiden Apostrophen darf weder ein Leerzeichen noch ein Satzende stehen. An beliebiger anderer Stelle innerhalb eines Textes hat das Ende eines Eingabesatzes keine Wirkung.

7.2.2 Leere Eingabefelder

Bei formatgesteuerter Eingabe wird in jedes Element der Eingabeliste ein neuer Wert geschrieben; insbesondere bewirken auch leere Eingabefelder Wertzuweisungen.
Anders ist das bei der listengesteuerten Eingabe. Hier bewirkt ein leeres Eingabefeld (null value), daß der Wert des entsprechenden Eingabeelements unverändert erhalten bleibt. Leere Eingabefelder können auf verschiedene Arten erzeugt werden, nämlich
(1) durch Kommas, die entweder unmittelbar aufeinanderfolgen oder nur durch Leerzeichen voneinander getrennt sind,
(2) durch ein Komma vor dem ersten Wert im ersten Satz, sowie
(3) durch einen Schrägstrich (slash).

Der Schrägstrich bewirkt, daß das Lesen beendet wird; Eingabeelemente, die während der Übertragung bislang noch keinen Wert erhalten haben, bleiben so unverändert. Der Schrägstrich hat damit die gleiche Wirkung, als wenn für jedes verbleibende Element der Eingabeliste ein Leerfeld angegeben wäre.

Beispiel: Ein Programm enthalte die Anweisungen

```
DIMENSION A(100)
READ *, A
```

Die Eingabesätze

```
⅋,⅋17,17⅋⅋1-15⅋⅋,,16E2,,,,,,
1.1/
```

bewirken die Wertzuweisungen A(2)=17, A(3)=17, A(4)=1.E-15, A(6)=1600. und A(12)=1.1. Alle anderen Elemente von A werden nicht verändert.

7.2.3 Wiederholungsfaktoren

Eingabewerte für listengesteuerte Umwandlung können mit Wiederholungsfaktoren versehen werden, und zwar in der Form

r*c bzw. r*

Hierin ist r eine vorzeichenlose, positive INTEGER-Konstante und c ein Eingabewert.
Die Form r*c hat die gleiche Wirkung wie r (aufeinanderfolgende) Werte c; die Form r* hat die gleiche Wirkung wie r (aufeinanderfolgende) Leerfelder.

Leerzeichen zwischen dem Wiederholungsfaktor und dem Stern sind ebenso unzulässig wie Leerzeichen zwischen dem Stern und dem Wert c. Man beachte: 2*½2 erzeugt nicht zwei aufeinanderfolgende Werte 2, sondern zwei leere Eingabefelder und den nachfolgenden Wert 2.

7.2.4 Listengesteuerte Ausgabe

Bei der listengesteuerten Ausgabe muß der Rechner je nach Typ des Ausgabewertes zwischen den Formatbeschreibern Iw, 1PEw.dEe, OPFw.d, Lw und A mit anlagenspezifischen Werten w, d und e auswählen.

Die Norm schreibt für die listengesteuerte Ausgabe vor:

(1) Numerische und logische Werte werden jeweils durch ein Komma und/oder (mindestens) ein Leerzeichen voneinander getrennt. Vor, zwischen und hinter Texte wird kein Trennzeichen gesetzt.

(2) Komplexe Werte werden in Klammern eingeschlossen; zwischen ihren Real- und Imaginärteil wird ein Komma gesetzt.

(3) Texte werden ohne begrenzende Apostrophe geschrieben. Ebenso werden in einem Text enthaltene Apostrophe nicht durch jeweils zwei Apostrophe dargestellt.

(4) Aufeinanderfolgende, identische Ausgabewerte c dürfen, müssen jedoch nicht, in der Form r*c mit einem Wiederholungsfaktor r geschrieben werden.

(5) Anlagenabhängig ist festzulegen, wann ein Satz abgeschlossen und der nächste begonnen wird. Dabei dürfen zwar Texte, nicht aber numerische Ausgabefelder durch ein Satzende unterbrochen werden. (Real- und Imaginärteil eines komplexen Wertes belegen jeweils ein eigenes Ausgabefeld!)

(6) Leere Ausgabefelder werden nicht erzeugt.

(7) Das erste Zeichen in jedem Satz ist ein Leerzeichen, das im Fall des Druckens einen einfachen Zeilenvorschub bewirkt. Bei Texten, die über ein Satzende hinaus fortgesetzt werden, wird dieses Leerzeichen in die Zeichenfolge des Textes eingefügt.

Man beachte: Eine Datei, die listengesteuert geschrieben wurde, kann dann und nur dann listengesteuert wieder gelesen werden, wenn nur numerische und logische Werte, nicht jedoch Texte geschrieben wurden.

7.3 Interne Dateien

Als interne Dateien können verwendet werden
(1) Textvariable,
(2) Textfelder,
(3) Elemente von Textfeldern, und
(4) Teilketten.

Für interne Dateien stehen nur die Anweisungen READ und WRITE zur Verfügung; der Zugriff auf eine interne Datei erfolgt durch die Angabe ihres Namens im Steuerparameter [UNIT=]u.

Zulässig sind nur formatierte Übertragungen, die außerdem grundsätzlich formatgesteuert erfolgen müssen.

Textvariable, Elemente von Textfeldern und Teilketten bestehen als interne Dateien jeweils aus genau einem Satz. Bei Textfeldern bildet dagegen jedes Element einen Satz; die Anzahl der Elemente eines Textfeldes, das als interne Datei angesprochen werden soll, muß in der jeweiligen Programmeinheit explizit festgelegt sein, d.h. in der Vereinbarung des Feldes darf zur Bestimmung der Obergrenze des Index der letzten Dimension kein Stern angegeben sein.

Anders als eine externe Datei wird eine interne Datei am Beginn jeder Übertragung erneut auf ihren Anfang positioniert, d.h. im Fall eines Textfeldes auf den Anfang des ersten Satzes, sonst auf den Anfang des einen Satzes, aus dem die Datei besteht. Die Änderungen der Positionierung während einer Übertragung erfolgen dagegen wie bei einer externen Datei.

Das Ende einer internen Datei besteht, anders als bei externen Dateien, aus dem Ende ihres (letzten) Satzes, und nicht aus einem besonderen EOF-Satz. Falls beim Lesen einer internen Datei deren Ende erreicht bzw. überschritten wird, erfolgen dieselben Aktionen wie bei externen Dateien; insbesondere haben die Steuerparameter IOSTAT=ios und END=s dieselbe Bedeutung wie bei externen Dateien (siehe 6.2.4).

Auf eine Textgröße, die an einer Stelle einer Programmeinheit als interne Datei angesprochen wird, kann an anderer Stelle der Programmeinheit ohne weiteres in anderer Form zugegriffen werden, z.B. als Operand einer Konkatenation oder als Element einer Ein-/Ausgabeliste.

Beispiel: Mit internen Dateien wird man zum einen dann arbeiten, wenn sich eine gewünschte externe Darstellung eines Wertes mit den vorhandenen Formatbeschreibern nicht direkt erzeugen läßt.
Es seien etwa DM-Beträge auszugeben. Dabei seien führende Leerzeichen durch Sterne und der Dezimalpunkt durch ein Komma zu ersetzen; außerdem soll auf jeden Fall ein Vorzeichen rechts neben die Ziffernfolge gesetzt werden, vor dem Komma mindestens eine Ziffer stehen.
Gelöst werden soll das Problem durch ein Funktionsunterprogramm mit dem umzuwandelnden (REAL-)Wert als erstem und der Länge der zu erzeugenden Zeichenfolge als zweitem Parameter:

```
CHARACTER*(*) FUNCTION UMWAND(WERT,L)
CHARACTER*10 FORM
FORM='(SP,F  .2)'
WRITE(FORM(6:7),'(I2)') L-1
WRITE(UMWAND,FORM) WERT
DO 1 I=1,L,1
```

```
      IF (UMWAND(I:I).NE.' ') GOTO 2
    1 UMWAND(I:I)='*'
    2 UMWAND(L:L)=UMWAND(I:I)
      UMWAND(I:I)='*'
      UMWAND(L-3:L-3)=','
      IF (UMWAND(L-4:L-4).EQ.'*') UMWAND(L-4:L-4)='0'
      END
```

Mit diesem Unterprogramm schreibt die Anweisungsfolge

```
      CHARACTER*500 UMWAND,TEXT(2)*8
      TEXT(1)=UMWAND(-17.5,8)
      TEXT(2)=UMWAND(.41,8)
      WRITE(19,'(1X,A,2X,A)') TEXT
```

in die Datei mit der Nummer 19 als einen Satz die Zeichenfolge

␣**17,50-␣␣***0,41+

Die Funktion UMWAND demonstriert gleichzeitig eine zweite Anwendung interner Dateien: Jedes Format muß die Formatbeschreiber in der externen Darstellung enthalten; falls ein Format nicht bereits beim Schreiben des Programms in allen Einzelheiten festgelegt werden kann, sondern von Größen abhängt, die erst im Programm berechnet werden, wird man zum Aufbau des Formats, zumindest teilweise, die Möglichkeiten der internen Dateien nutzen. In der Funktion UMWAND wird so die jeweilige Größe des Ausgabefeldes in das Format eingesetzt.

8 Speicherplatzorganisation und Anfangswerte

In diesem Abschnitt werden nicht ausführbare Anweisungen behandelt, für die in erster Linie in umfangreichen Programmen Bedarf besteht.

8.1 COMMON

Jede Programmeinheit hat einen eigenen Speicherbereich, der von anderen Programmeinheiten nicht direkt beeinflußt werden kann. Über Parameterlisten können Daten zwischen zwei Programmeinheiten übergeben werden.

Es gibt eine zweite Methode der Datenübergabe: Man kann Speicherbereiche definieren, genannt COMMON-Blöcke, auf die von mehreren Programmeinheiten zugegriffen werden kann. Dieses geschieht durch die Anweisung

COMMON [/[cb]/]nliste[[,]/[cb]/nliste]...

Hierin sind

cb Namen von COMMON-Blöcken
nliste Namenlisten, die Namen von Variablen und Feldern sowie Felddefinitionen enthalten

Mit COMMON-Anweisungen können die gemeinsamen Speicherbereiche für verschiedene Programmeinheiten eines ausführbaren Programms vereinbart werden. Diese sind der unbenannte COMMON-Block und die benannten COMMON-Blöcke. Der unbenannte COMMON-Block wird angesprochen, sooft in der COMMON-Anweisung die Angabe eines Namens cb weggelassen wird. Aus der COMMON-Anweisung ist zu ersehen, daß das erste Paar Schrägstriche weggelassen werden kann, wenn der entsprechende Name cb fehlt. Die Namen der COMMON-Blöcke sind global, die in den Namenlisten erscheinenden Namen sind lokal. Der Name eines COMMON-Blocks kann auch eine lokale Größe in derselben Programmeinheit bezeichnen mit Ausnahme einer Konstanten, einer inneren Standardfunktion oder einer Variablen, deren Name mit dem einer externen Funktion identisch ist. Der Name eines COMMON-Blocks kann nur in einer COMMON- oder in einer SAVE-Anweisung (siehe 8.5) erscheinen. Felder

können in einer COMMON-Anweisung dimensioniert werden, wobei als Indexgrenzen nur konstante INTEGER-Ausdrücke zugelassen sind. Beispiel:
Statt

```
DIMENSION B(4,4)
COMMON B
```

kann es kürzer heißen

```
COMMON B(4,4)
```

Die Speicherplätze in einem COMMON-Block werden in der Reihenfolge an Variable und Felder vergeben, wie deren Namen in den Listen nliste erscheinen.

Bei byteorientierten Maschinen können numerische Datenelemente typabhängig auf Byte-, Halbwort-, Wort- oder Doppelwortgrenzen ausgerichtet gespeichert werden. Daher können in COMMON-Blöcken Lücken entstehen. Man kann eine lückenlose Belegung erreichen, wenn man die Datenelemente nach der Länge absteigend geordnet in den COMMON-Blöcken speichert, d.h. die längsten Datenelemente zuerst. Jeder COMMON-Block beginnt an einer Doppelwortgrenze.

Der Name eines COMMON-Blocks kann in einer COMMON-Anweisung mehrfach genannt werden. Wenn das geschieht, setzt die neue nliste die vorangehende nliste fort, die zu demselben COMMON-Block gehört. Derselbe COMMON-Block-Name kann auch mehrfach in verschiedenen COMMON-Anweisungen derselben Programmeinheit erscheinen. Dann werden die zugehörigen Namenlisten als aneinandergekettet aufgefaßt. Entsprechendes gilt für den unbenannten COMMON-Block. Beispiel: Die Anweisung

```
COMMON /BLK/N,A(10,10)//U,V,W/BLK/B(10),X(10)
```

ist äquivalent den Anweisungen

```
COMMON U,V,W
COMMON /BLK/N,A(10,10)
COMMON /BLK/B(10),X(10)
```

Die Listen eines COMMON-Blocks in COMMON-Anweisungen verschiedener Programmeinheiten werden auf den Anfang des gemeinsamen COMMON-

Blocks ausgerichtet.

Wenn ein Name in einer nliste mit dem Typ CHARACTER verbunden ist, müssen es auch alle anderen Namen der Liste sein. Auch in anderen Programmeinheiten dürfen dann nur Größen des Typs CHARACTER in den entsprechenden COMMON-Block gebracht werden.

Wenn zwischen zwei Programmeinheiten Daten übergeben werden sollen, ist auch sonst nur die Zuordnung von Größen gleichen Typs sinnvoll, abgesehen von den Typen REAL und COMPLEX. Da im Fall des Typs COMPLEX in zwei konsekutiven Worten REAL-Konstanten für den Real- und Imaginärteil einer komplexen Zahl dargestellt sind, kann es zweckmäßig sein, über COMMON-Anweisungen COMPLEX- und REAL-Größen einander zuzuordnen.

Stehe beispielsweise in Programmeinheit 1

```
COMPLEX Z(100)
COMMON Z
```

in Programmeinheit 2

```
REAL X(2,100)
COMMON X
```

so greifen in Programmeinheit 2 X(1,1), X(1,2),..., X(1,100) auf die Realteile und X(2,1), X(2,2),..., X(2,100) auf die Imaginärteile von Z(1), Z(2),..., Z(100) zu.

Es ist aber auch erlaubt, in einem COMMON-Block Daten der Typen INTEGER, DOUBLE PRECISION, COMPLEX oder LOGICAL einander zuzuordnen. Allerdings wird davon selten und nur in umfangreichen Programmen in dem Fall Gebrauch gemacht, daß Daten eines COMMON-Blocks nicht mehr benötigt werden und der COMMON-Block zur Einsparung von Speicherplatz auch von Daten anderen Typs belegt wird. Die Länge eines COMMON-Blocks, dessen Elemente vom Typ CHARACTER sind, wird durch die Anzahl seiner Zeichen gemessen, bei den anderen Typen durch die Anzahl seiner Worte. Dabei wird eine durch eine EQUIVALENCE-Anweisung (siehe 8.2) bewirkte Verlängerung eines COMMON-Blocks berücksichtigt.

Ein benannter COMMON-Block muß in allen Programmeinheiten, in denen

er genannt wird, mit der gleichen Länge vereinbart sein. Für den unbenannten COMMON-Block besteht diese Forderung nicht.

Somit kann das Beispiel zur Suche des Maximums und des Minimums in einem Feld aus Abschnitt 4.3.2 abgeändert werden zu

```
      COMMON B(4,4)
      READ*,B
      CALL MINMAX
      END
      SUBROUTINE MINMAX
      COMMON A(16)
      ...
      END
```

Benannte COMMON-Blöcke können nützlich sein, wenn ein Programm aus vielen Programmeinheiten besteht und diese jeweils nur einen Teil der in COMMON-Blöcken gespeicherten Daten benötigen. Dann können die von einer Programmeinheit benötigten Daten in einem Block zusammengefaßt werden, die übrigen brauchen in dieser Programmeinheit nicht genannt zu werden. Allerdings kann man die gleiche Wirkung durch Datenübergabe mit Parameterlisten erzielen. Als geringfügiger Vorteil der Übergabe mit COMMON-Anweisungen bleibt, daß Unterprogrammaufrufe kürzer werden.

Beispiel für die Anwendung benannter COMMON-Blöcke:

Programmeinheit 1

```
      COMPLEX U,V,W
      COMMON/B1/R,S,X(5)/B2/U,V,W/B3/A(6,6),B(5)
```

Programmeinheit 2

```
      COMPLEX R,S
      COMMON/B2/R,S,X,Y
```

Programmeinheit 3

```
      COMMON/B3/P(5,5),Q(16)
```

Nach Ausführung der Anweisung END oder RETURN in einem Unterprogramm können Daten in einem benannten COMMON-Block durch Überspeicherung in einen undefinierten Zustand übergehen. Das kann jedoch

nur dann eintreten, wenn der COMMON-Block in der Programmeinheit, die das Unterprogramm direkt oder indirekt aufrief, nicht genannt ist. Diese Daten können durch die Anweisung SAVE (siehe 8.5) gerettet werden.

Ob der genannte Fall überhaupt eintreten kann, hängt von der jeweiligen Anlage ab.

8.2 EQUIVALENCE

Während mit der COMMON-Anweisung Variablen und Feldern aus verschiedenen Programmeinheiten dieselben Speicherplätze zugewiesen werden, wirkt die Anweisung

EQUIVALENCE (n,n[,n]...)[,(n,n[,n]...)]...

auf Variablen und Felder derselben Programmeinheit und weist ihnen dieselben Speicherplätze zu. Es bedeuten

n Variablen, Felder ,Feldelemente oder Teilketten

Es ist klar, daß jede Liste n,n[,n]... mindestens zwei Elemente enthalten muß. Alle Elemente werden auf einen gemeinsamen Anfang ausgerichtet.

Ein Listenelement vom Typ CHARACTER darf nur mit einem anderen vom Typ CHARACTER in einer EQUVALENCE-Anweisung gleichgesetzt werden. Es wird dann das erste Zeichen dieser Elemente auf denselben Platz gesetzt, ihre Längen dürfen verschieden sein.

Sonst dürfen auch Listenelemente verschiedenen Typs gleichgesetzt werden. Eine Typumwandlung ist damit nicht verbunden.

Wenn der Name eines Feldes in einer EQUIVALENCE-Anweisung genannt wird, so wird das erste Feldelement angesprochen.

Als Indizes in Feldelementen und als erste und letzte Zeichenposition von Teilketten dürfen nur konstante INTEGER-Ausdrücke vorkommen. Bekanntlich sind Feldelemente in aufeinanderfolgenden Plätzen gespeichert. Werden in einer Liste zwei Elemente zweier Felder genannt, so erhalten daher nicht nur diese gleichen Spei-

cherplatz, sondern ggf. auch weitere Elemente.

Beispiel:
```
DIMENSION A(-2:2),B(4)
EQUIVALENCE (A(2),B(3),X)
```

bewirkt folgende Speicherbelegung

```
A(-2)
A(-1)
A(0)      B(1)
A(1)      B(2)
A(2)      B(3)      X
          B(4)
```

Feldelemente und Variablen, die denselben Speicherplatz belegen, sind in derselben Zeile notiert.

Entsprechend können auch mehrfach indizierte Feldelemente anderen Listenelementen gleichgesetzt werden.

Selbstverständlich ist es unzulässig, zwei Namen aus COMMON-Anweisungen oder zwei Elemente desselben Feldes gleichzusetzen. Unzulässig ist es aber auch, indirekt eine verbotene Gleichsetzung vorzunehmen wie in der Anweisung

```
EQUIVALENCE (A(1),B(1)),(A(1),B(100))
```

Es ist aber möglich, durch eine EQUIVALENCE-Anweisung eine nicht in einer COMMON-Anweisung genannte Variable oder ein Feld in einen COMMON-Block hineinzubringen, wobei er auch "über sein Ende hinaus" verlängert werden kann.

Beispiel:
```
COMMON/X/A(3),Z
DIMENSION B(5)
EQUIVALENCE (A(3),B(2))
```

Ohne die EQUIVALENCE-Anweisung würde der COMMON-Block X 4 Plätze umfassen. Durch die EQUIVALENCE-Anweisung wird er über sein Ende hinaus verlängert, woraus folgende Speicherplatzverteilung resultiert:

```
A(1)
A(2)        B(1)
A(3)        B(2)
Z           B(3)
            B(4)
            B(5)
```

Eine entsprechende Verlängerung eines COMMON-Blocks "über seinen Anfang hinaus" ist unzulässig.

Als Anwendungsbeispiel für die EQUIVALENCE-Anweisung wird das Programm zur Suche des Minimums und des Maximums eines Feldes aus den Abschnitten 4.3.2 und 8.1 nochmals etwas abgewandelt.

```
      DIMENSION A(4,4),B(16)
      EQUIVALENCE (A(1,1),B(1))
*     MAN HAETTE AUCH EQUIVALENCE (A,B) SCHREIBEN KOENNEN
      READ*,A
      (Hier folgt der Algorithmus zur Suche des Minimums
      und des Maximums unmittelbar, nicht als Unterpro-
      gramm formuliert)
      END
```

8.3 DATA

Bei Beginn der Ausführung eines Programms sind die Werte aller darin genannten Variablen, Felder und Teilketten undefiniert. Bisher wurden diesen Größen Werte in Eingabeanweisungen und Wertzuweisungen während der Ausführung eines Programms erteilt. Vor oder zu Beginn der Ausführung eines Programms können allen Größen Anfangswerte durch die Anweisung

```
DATA nliste/cliste/[[,]nliste/cliste/]...
```

erteilt werden. Hierin sind

nliste Liste von Variablen, Feldelementen, Feldern oder Teilketten sowie impliziten DO-Elementen

cliste Konstantenliste a[,a].... Dabei hat a die Form c, wo c Konstante oder Name einer Konstanten ist, oder

r*c, wo r eine von Null verschiedene vorzeichenlose INTEGER-Konstante oder der Name einer solchen Konstanten ist und einen Wiederholungsfaktor bedeutet.

Die Namenliste entspricht einer Eingabeliste, jedoch dürfen Größen aus dem unbenannten COMMON-Block keine Anfangswerte erteilt werden, Größen aus einem benannten COMMON-Block nur in einem BLOCK DATA-Unterprogramm (siehe 8.4).

Es muß eine eins-zu-eins-Zuordnung zwischen den Größen einer Namenliste und der zugehörigen Konstantenliste existieren. Wenn der Name eines Feldes genannt wird, muß jedes Element mit einem Wert besetzt werden, wobei die Zuordnung durch die Speicherabbildungsfunktion und die Reihenfolge der Konstanten in der Konstantenliste bestimmt ist.

Falls eine durch die Namenliste spezifizierte Zielgröße und ihre entsprechende Konstante von verschiedenen arithmetischen Typen sind, erfolgt Typumwandlung wie bei einer arithmetischen Wertzuweisung. Im Falle der Typen LOGICAL und CHARACTER muß Typgleichheit für Zielgröße und Konstante bestehen. Wenn beide vom Typ CHARACTER und von unterschiedlicher Länge sind, so wird wie bei der Wertzuweisung für Texte verfahren, d.h. wenn die Konstante kürzer als die Zielgröße ist, so wird die Konstante linksbündig übertragen und rechts wird mit Leerzeichen aufgefüllt; wenn die Konstante länger als die Zielgröße ist, so werden in der Konstanten rechts stehende Zeichen ignoriert.

Einer Größe darf nur einmal in einem ausführbaren Programm ein Anfangswert erteilt werden. Das gilt entsprechend, wenn zwei Größen gleichen Speicherplatz belegen. Namen von Funktionen und Eingangspunkten darf kein Anfangswert zugewiesen werden.

Die Positionen des ersten oder letzten Zeichens von Teilketten müssen als konstante INTEGER-Ausdrücke angegeben werden. Bei Feldelementen dürfen in den Indexausdrücken zusätzlich auch Laufvariablen von impliziten DO-Elementen erscheinen.

Implizite DO-Elemente (doliste,i=e_1,e_1[,e_3]) sind nach Regeln zu bilden, die im Vergleich zu den in Abschnitt 6.2.5 genannten etwas

einschränkend sind. Die Laufvariable i muß vom Typ INTEGER sein, e_1, e_2, e_3 müssen konstante INTEGER-Ausdrücke sein mit der Ausnahme, daß Laufvariable übergeordneter impliziter DO-Elemente als Operanden verwendet werden dürfen. Der Durchlaufzähler muß positiv sein. Die Laufvariable ist lokal bezüglich des DO-Elements, d.h. sie kann sonst beliebig zur Bezeichnung anderer Größen verwendet werden. Beispiel:

```
      DIMENSION D(5)
      DOUBLE PRECISION A(10,10),X(10)
      CHARACTER*15 TEXT
      COMPLEX C
      LOGICAL R,S
      DATA X,(A(I,I),I=1,10),D(3)/20*1.,.5/C,
     *TEXT/(1.,1.),'ANFANGSWERTE'/R,S/.TRUE.,
     *.FALSE./
```

Bemerkung: Die DATA-Anweisung erteilt TEXT den Anfangswert ANFANGSWERTEbbb.

Wenn die Werte des Beispiels während der Ausführung eines Programms zugewiesen werden sollen, so kann das durch die Anweisungen

```
      DO 10 I=1,10
      X(I)=1.
10    A(I,I)=1.
      D(3)=.5
      C=(1.,1.)
      TEXT='ANFANGSWERTE'
      R=.TRUE.
      S=.FALSE.
```

geschehen.

8.4 BLOCK DATA

An Variable und Felder in benannten COMMON-Blöcke dürfen Anfangswerte nur in BLOCK DATA-Unterprogrammen zugewiesen werden, an Variable und Felder im unbenannten COMMON-Block überhaupt nicht (siehe 8.1).

Die erste Anweisung eines BLOCK DATA-Unterprogramms ist

BLOCK DATA [name]

wo name der Name des BLOCK DATA-Unterprogramms ist. Sonst dürfen nur die Anweisungen IMPLICIT, PARAMETER, DIMENSION, COMMON, SAVE (siehe 8.5), EQUIVALENCE, DATA, Typanweisungen und END in einem BLOCK DATA-Unterprogramm vorkommen. Kommentarzeilen sind natürlich erlaubt.

Ein BLOCK DATA-Unterprogramm dient nur der Erteilung von Anfangswerten an Variablen und Felder in benannten COMMON-Blöcken. Es ist keine Prozedur und wird nirgends aufgerufen. Die Benennung von BLOCK DATA-Unterprogrammen ist nur erforderlich, wenn mehrere davon vorhanden sind. Die Nennung eines BLOCK DATA-Unterprogramms in einer EXTERNAL-Anweisung kann nur die Information beinhalten, daß dieses Unterprogramm an ein ausführbares Programm anzuschließen ist.

Auch für BLOCK DATA-Unterprogramme gilt, daß benannte COMMON-Blöcke dieselbe Länge wie in den anderen Programmeinheiten haben müssen.

Wenn die Variablen und Felder des Beispiels aus Abschnitt 8.3 in COMMON-Blöcken stehen, können die Anfangswerte durch das folgende BLOCK DATA-Unterprogramm erteilt werden.

```
      BLOCK DATA
      DOUBLE PRECISION A,X
      CHARACTER*15 TEXT
      COMPLEX C
      LOGICAL R,S
      COMMON/BL1/X(10),A(10,10),D(5)/BL2/C,R,S/BL3/TEXT
      DATA X,A(I,I),I=1,10),D(3)/20*1.,.5/,TEXT/
     *,'ANFANGSWERTE'/,C,R,S/(1.,1.),.TRUE.,.FALSE./
      END
```

8.5 SAVE

Nach der Ausführung einer RETURN- oder END-Anweisung in einer externen Prozedur gehen Hilfsgrößen und u.U. in einem benannten

COMMON-Block stehende Daten, weil sie alle lokal sind, in einen undefinierten Zustand über. Sie können nämlich durch andere Daten überspeichert werden und stehen damit beim nächsten Aufruf nicht mehr zur Verfügung. So sieht es jedenfalls die Norm vor. Ob der Fall tatsächlich eintreten kann, hängt von der betreffenden Anlage ab. Es ist nämlich auch möglich, daß vor der Ausführung ein Maschinenprogramm erzeugt wird, das alle Unterprogramme mit festen Speicherplätzen enthält. Dann wird auch nichts überspeichert. Die in der Norm beschriebene Situation tritt ein, wenn während der Ausführung eines Programms Unterprogramme dynamisch nachgeladen werden.

Lokale Größen können ggf. durch die Anweisung

```
SAVE [a,[a]...]
```

auf zusätzlich bereitgestellte Speicherplätze gerettet werden. Darin bezeichnen

a Variablen, Felder oder
COMMON-Blöcke, deren Namen in Schrägstriche eingeschlossen sind

Sind keine Listenelemente in der Anweisung SAVE angegeben, so werden alle lokalen Größen des Unterprogramms gerettet.

Formale Parameter dürfen natürlich nicht in einer SAVE-Anweisung genannt werden, auch nicht Größen, die in einem COMMON-Block erscheinen. Wenn ein COMMON-Block in Schrägstrichen in einer SAVE-Anweisung genannt ist, werden alle Größen des COMMON-Blocks erfaßt.

Ein simples Anwendungsbeispiel ergibt die Aufgabe, in einer Hilfsvariablen die Aufrufe eines Unterprogramms zu zählen.

```
PROGRAM HAUPT
...
CALL UP
...
END
SUBROUTINE UP
SAVE I
```

```
DATA I/0/
I=I+1
...
END
```

Die SAVE-Anweisung beläßt also nach Verlassen von UP die Hilfsvariable I in definiertem Zustand. Ohne SAVE gibt es zwei Möglichkeiten: Die Hilfsgrössen können durch Überspeicherung in einen undefinierten Zustand übergehen oder sie bleiben stets auch ohne SAVE-Anweisung erhalten. Oben wurde darauf hingewiesen, daß es von der betreffenden Anlage abhängt, welcher der beiden Fälle eintritt.

Wenn ein COMMON-Block in einer SAVE-Anweisung eines Unterprogramms genannnt ist, dann muß er auch in allen anderen Unterprogrammen, in denen er in einer COMMON-Anweisung steht, in einer SAVE-Anweisung genannt werden.

Nicht betroffen von unkontrollierbarer Überspeicherung sind

(1) benannte COMMON-Blöcke, die auch in der rufenden Programmeinheit genannt sind,
(2) der unbenannte COMMON-Block,
(3) durch DATA-Anweisung definierte Größen, solange sie nicht verändert wurden.

Daß die unter (3) genannten Größen erhalten bleiben, folgt aus den am Anfang dieses Abschnitts gemachten Bemerkungen. Entweder haben alle Größen feste Speicherplätze; oder während der Ausführung eines Programms werden Unterprogramme dynamisch nachgeladen. Wenn das der Fall ist, werden bei jedem Nachladen die Anfangswerte neu erteilt.

Anhang A: ASCII- und EBCDIC-Code

Die Tabelle gibt die druckbaren Zeichen des (128-Zeichen-)ASCII-Code und des EBCDIC-Code an. Zu jedem Zeichen ist der Wert eingetragen, den die innere Standardfunktion ICHAR mit ihm als Parameter liefert (siehe Anhang B.4). Für andere Codes, die in der Praxis verwendet werden, sei auf die jeweiligen Benutzer-Handbücher verwiesen.

Sowohl der (128-Zeichen-)ASCII-Code als auch der EBCDIC-Code enthalten neben den druckbaren Zeichen Steuerzeichen, die nicht druckbar sind; diese sind in die Tabelle nicht aufgenommen.

Ein Strich anstelle einer Zahl kennzeichnet, daß es dieses Zeichen im jeweiligen Code nicht gibt.

Man beachte: Nicht jedes Ausgabegerät verfügt über den vollen ASCII- bzw. EBCDIC-Zeichensatz. Außerdem werden teilweise bei Ausgabegeräten einzelne Sonderzeichen durch andere ersetzt, die im eigentlichen Zeichensatz nicht enthalten sind.

Zeichen	ASCII	EBCDIC	Zeichen	ASCII	EDCDIC
␣	32	64	/	47	97
!	33	90	0	48	240
"	34	127	1	49	241
#	35	123	2	50	242
$	36	91	3	51	243
%	37	108	4	52	244
&	38	80	5	53	245
'(Apostroph)	39	125	6	54	246
(	40	77	7	55	247
)	41	93	8	56	248
*	42	92	9	57	249
+	43	78	:	58	122
,(Komma)	44	107	;	59	94
-(Minus)	45	96	<	60	76
.(Punkt)	46	75	=	61	126

Zeichen	ASCII	EBCDIC
>	62	110
?	63	111
@	64	124
A	65	193
B	66	194
C	67	195
D	68	196
E	69	197
F	70	198
G	71	199
H	72	200
I	73	201
J	74	209
K	75	210
L	76	211
M	77	212
N	78	213
O	79	214
P	80	215
Q	81	216
R	82	217
S	83	226
T	84	227
U	85	228
V	86	229
W	87	230
X	88	231
Y	89	232
Z	90	233
[	91	-
\	92	224
]	93	-
^	94	-
_ (Unterstreichung)	95	109

Zeichen	ASCII	EBCDIC
`(Akzent)	96	121
a	97	129
b	98	130
c	99	131
d	100	132
e	101	133
f	102	134
g	103	135
h	104	136
i	105	137
j	106	145
k	107	146
l	108	147
m	109	148
n	110	149
o	111	150
p	112	151
q	113	152
r	114	153
s	115	162
t	116	163
u	117	164
v	118	165
w	119	166
x	120	167
y	121	168
z	122	169
{	123	192
|	124	106
}	125	208
~	126	161
V(log. Oder)	-	79
¢	-	74
¬(nicht)	-	95

Anhang B: Innere Standardfunktionen

B.1 Funktionen zur Typumwandlung

Vorbemerkungen:

(1) Da sich die Typen der Funktionswerte unmittelbar aus der Bedeutung der Funktionen ergeben, sind sie in der Tabelle nicht besonders aufgeführt.

(2) Grundsätzlich muß der Wert eines Ausdrucks, der als Parameter angegeben wird, innerhalb des Zahlenbereiches liegen, der für den jeweiligen Funktionswert zur Verfügung steht.

(3) Die Namen aller Funktionen zur Typumwandlung dürfen nicht als aktuelle Parameter von Unterprogrammaufrufen verwendet werden.

	Anzahl der Parameter	Gattungsname	Spezieller Name	Parametertyp
Umwandlung in INTEGER (1),(2)	1	INT	- INT IFIX IDINT -	INTEGER REAL REAL DOUBLE COMPLEX
Umwandlung in REAL (3),(4)	1	REAL	REAL FLOAT - SNGL -	INTEGER INTEGER REAL DOUBLE COMPLEX
		-	AIMAG	COMPLEX
Umwandlung in DOUBLE PRECISION (2)	1	DBLE	- - - -	INTEGER REAL DOUBLE COMPLEX
Umwandlung in COMPLEX (4),(5)	1 oder 2	CMPLX	- - - -	INTEGER REAL DOUBLE COMPLEX

Bemerkungen:

(1) Umwandlung in den Typ INTEGER besteht im Abschneiden der Stellen hinter dem Dezimalpunkt. Eine Rundung erfolgt dabei nicht (siehe auch Anhang B.2).
Die Funktionen mit den speziellen Namen INT und IFIX haben dieselbe Wirkung.

(2) Bei einem komplexen Parameter liefert die Funktion als Resultat den umgewandelten Realteil.

(3) Die Funktionen mit den speziellen Namen REAL und FLOAT haben dieselbe Wirkung.
Bei einem komplexen Parameter liefert REAL den Realteil und AIMAG den Imaginärteil als Funktionswert.

(4) Bei der Umwandlung kann Genauigkeit verloren gehen: Die maximale Stellenzahl von INTEGER-Größen ist i.a. größer als die Anzahl der signifikanten Stellen von REAL- bzw. COMPLEX-Größen; die Mantisse einer DOUBLE PRECISION-Größe besteht i.a. aus mehr Ziffern als die Mantisse einer REAL- bzw. COMPLEX-Größe.

(5) Falls die Funktion CMPLX mit einem Parameter aufgerufen wird, ergibt dessen Wert den Realteil des Funktionswertes; der Imaginärteil des Funktionswertes erhält den Wert Null.
Falls die Funktion CMPLX mit zwei Parametern aufgerufen wird, ergibt der Wert des ersten Parameters den Realteil und der Wert des zweiten Parameters den Imaginärteil des Funktionswertes; beide Parameter müssen hier denselben Typ besitzen.

B.2 Arithmetische Funktionen

Vorbemerkungen:

(1) Bei den Funktionen mit mehreren Parametern müssen alle Parameter denselben Typ besitzen.

(2) Bei allen Funktionen, für die die Rubrik "Typ" nur eine Angabe enthält, besitzt der Funktionswert denselben Typ wie der bzw. die Parameter. Bei den übrigen Funktionen bezieht sich in der Rubrik "Typ" die erste Angabe auf die Parameter und die zweite auf den Funktionswert.

	Anzahl der Parameter	Gattungsname	Spezieller Name	Typ
Abschneiden hinter dem Dezimalpunkt	1	AINT	AINT DINT	REAL DOUBLE
Rundung (1)	1	ANINT	ANINT DNINT	REAL DOUBLE
		NINT	NINT IDNINT	REAL/INT DOUBLE/INT
Absolutbetrag	1	ABS	IABS ABS DABS CABS	INTEGER REAL DOUBLE CMPLX/REAL
Divisionsrest (2)	2	MOD	MOD AMOD DMOD	INTEGER REAL DOUBLE
Vorzeichenübertragung (3)	2	SIGN	ISIGN SIGN DSIGN	INTEGER REAL DOUBLE
Positive Differenz (4)	2	DIM	IDIM DIM DDIM	INTEGER REAL DOUBLE
Maximum (5)	≥ 2	MAX	MAXO AMAX1 DMAX1	INTEGER REAL DOUBLE
		-	AMAXO MAX1	INT/REAL REAL/INT

	Anzahl der Parameter	Gattungs-name	Spezieller Name	Typ
Minimum (5)	≥ 2	MIN	MINO AMIN1 DMIN1	INTEGER REAL DOUBLE
		-	AMINO MIN1	INT/REAL REAL/INT
Produkt (6)	2	-	DPROD	REAL/DOUBLE
Konjugiert Komplexe	1	-	CONJG	COMPLEX
Quadratwurzel (7)	1	SQRT	SQRT DSQRT CSQRT	REAL DOUBLE COMPLEX

Bemerkungen:

(1) Berechnet wird (A)NINT(a)=(A)INT(a+.5) im Fall $a \geq 0$ und (A)NINT(a)=(A)INT(a-.5) im Fall $a<0$.

(2) Berechnet wird MOD(a,b)=a-INT(a/b)*b. Dabei muß $b \neq 0$ gelten.

(3) Geliefert wird SIGN(a,b)=|a| im Fall $b \geq 0$ und SIGN(a,b)=-|a| im Fall $b<0$.

(4) Geliefert wird DIM(a,b)=MAX(a-b,0).

(5) Die Funktionen dürfen nicht als aktuelle Parameter in Unterprogrammaufrufen verwendet werden.

(6) Bei der Multiplikation von zwei REAL-Größen wird das Resultat zunächst mit voller Genauigkeit ausgerechnet. Während bei Verwendung des Multiplikationsoperators * die überzähligen Stellen anschließend abgeschnitten bzw. gerundet werden, liefert die Funktion DPROD das Resultat in voller Länge.

(7) Bei einem reellen Parameter wird immer ein reeller Funktionswert geliefert; entsprechend müssen reelle Parameter immer nichtnegativ sein.
Bei einem komplexen Parameter wird der (komplexe) Hauptwert geliefert.

B.3 Mathematische Funktionen

Vorbemerkung: Die Funktionswerte aller mathematischen Funktionen besitzen denselben Typ wie der bzw. die Parameter. Deshalb wird in der Tabelle jeweils nur pauschal der Typ angegeben.

	Anzahl der Parameter	Gattungs-name	Spezieller Name	Typ
Exponentialfunktion: e**x	1	EXP	EXP DEXP CEXP	REAL DOUBLE COMPLEX
Natürlicher Logarithmus: ln x (1)	1	LOG	ALOG DLOG CLOG	REAL DOUBLE COMPLEX
Dekadischer Logarithmus: lg x (1)	1	LOG10	ALOG10 DLOG10	REAL DOUBLE
Sinus: sin x (2)	1	SIN	SIN DSIN CSIN	REAL DOUBLE COMPLEX
Kosinus: cos x (2)	1	COS	COS DCOS CCOS	REAL DOUBLE COMPLEX
Tangens: tan x (2)	1	TAN	TAN DTAN	REAL DOUBLE
Arkussinus: arcsin x (2),(3)	1	ASIN	ASIN DASIN	REAL DOUBLE
Arkuskosinus: arccos x (2),(3)	1	ACOS	ACOS DACOS	REAL DOUBLE
Arkustangens: arctan x (2),(4)	1	ATAN	ATAN DATAN	REAL DOUBLE
Arkustangens: arctan x/y (2),(5)	2	ATAN2	ATAN2 DATAN2	REAL DOUBLE
Hyperbelsinus: Sinh x	1	SINH	SINH DSINH	REAL DOUBLE
Hyperbelkosinus: Cosh x	1	COSH	COSH DCOSH	REAL DOUBLE
Hyperbeltangens: Tanh x	1	TANH	TANH DTANH	REAL DOUBLE

Bemerkungen:

(1) Reelle Parameter müssen positive Werte und komplexe Parameter Werte ungleich Null besitzen.
Für den Imaginärteil i eines komplexen Resultats gilt

$$-\pi < i \leq \pi$$

(2) Als Parameter der trigonometrischen Funktionen müssen Werte im Bogenmaß angegeben werden. Die Umkehrfunktionen liefern Werte im Bogenmaß.

(3) Für den Parameter x muß gelten $|x| \leq 1$. Für die Funktionswerte gilt

$$-\pi/2 \leq \mathrm{ASIN}(x) \leq \pi/2$$
$$0 \leq \mathrm{ACOS}(x) \leq \pi$$

(4) Für den Funktionswert gilt

$$-\pi/2 \leq \mathrm{ATAN}(x) \leq \pi/2$$

(5) Für den Funktionswert gilt generell

$$-\pi < \mathrm{ATAN}(x,y) \leq \pi$$

Im einzelnen:

ATAN(x,y) < 0	im Fall	x < 0
= - π/2	im Fall	x < 0 , y = 0
= 0	im Fall	x = 0 , y > 0
= π/2	im Fall	x > 0 , y = 0
= π	im Fall	x = 0 , y < 0
> 0	im Fall	x > 0

Beide Parameter dürfen nicht gleichzeitig den Wert Null besitzen; ihr Typ muß übereinstimmen.

B.4 Funktionen zur Textverarbeitung

Vorbemerkung: Für die Funktionen zur Textverarbeitung gibt es keine Gattungsnamen. Die angegebenen Namen sind also sämtlich spezielle Namen.

	Anzahl der Parameter	Name	Parametertyp	Funktionswert
Position eines Zeichens in der Sortierfolge (1)	1	ICHAR	CHAR*1	INTEGER
Zeichen an einer Position in der Sortierfolge (1)	1	CHAR	INTEGER	CHAR*1
Länge eines Textausdrucks (2)	1	LEN	CHAR*(*)	INTEGER
Index eines Textes (3)	2	INDEX	CHAR*(*)	INTEGER
Vergleich von Texten (4)	2	LGE LGT LLE LLT	CHAR*(*) CHAR*(*) CHAR*(*) CHAR*(*)	LOGICAL LOGICAL LOGICAL LOGICAL

Bemerkungen:

(1) ICHAR liefert zu einem Textausdruck mit der Länge 1 die Position des Zeichens in der Sortierfolge des Rechners. Man beachte: In der Sortierfolge beginnt die Numerierung mit Null und endet mit n-1, wenn der verwendete Code n Zeichen umfaßt.
Umgekehrt liefert CHAR zu einem INTEGER-Wert als Parameter das Zeichen, das an der entsprechenden Stelle in der Sortierfolge des Rechners steht. Zulässig sind als Parameter nur Werte i mit $0 \leq i \leq n-1$, wo n die Anzahl der Zeichen des verwendeten Code ist.

Beide Funktionen sind nicht auf druckbare Zeichen beschränkt; bei der Anzahl der Zeichen eines Code sind in jedem Fall die (nicht druckbaren) Steuerzeichen mitzuzählen.

Beide Funktionen dürfen nicht als aktuelle Parameter im Aufruf von Unterprogrammen verwendet werden.

(2) Textvariable, Elemente von Textfeldern und Teilketten können auch dann als Parameter angegeben werden, wenn ihnen noch kein Wert zugewiesen ist.
Das Resultat von LEN(c) ist die Summe der Längen, mit denen die Operanden von c vereinbart sind.

(3) INDEX(c,d) liefert den Wert n, wenn an der Zeichenposition n in c eine Teilkette beginnt, die mit d übereinstimmt. Bei mehreren Übereinstimmungen wird die Position der (von links) ersten geliefert; das Resultat ist Null, wenn d nicht als Teilkette in c vorkommt, also insbesondere dann, wenn d länger ist als c.

Beispiele: Es gilt

```
INDEX('ABCABC','BC')=2
INDEX('BC','ABCABC')=0
INDEX('ABCABC','BD')=0
```

(4) Die Vergleichsfunktionen liefern den Wert "wahr", wenn ihre Parameter den folgenden Relationen genügen:

LGE(c,d) :	$c \geq d$
LGT(c,d) :	$c > d$
LLE(c,d) :	$c \leq d$
LLT(c,d) :	$c < d$

Alle vier Funktionen arbeiten auf der Basis der Sortierfolge des ASCII-Code. Ihr Resultat ist undefiniert, wenn einer der beiden Parameter ein Zeichen umfaßt, das im ASCII-Code nicht enthalten ist (siehe auch 5.2.3).
Alle vier Funktionen dürfen nicht als Parameter im Aufruf von Unterprogrammen verwendet werden.

Anhang C: Restriktionen des Subset

Vorbemerkung: Die Einschränkungen, die zu beachten sind, wenn nur der Subset und nicht der volle Sprachumfang zur Verfügung steht, werden hier stichwortartig wiedergegeben. Das Fehlen eines einzelnen Sprachelements hat zumeist weitere Restriktionen für andere Sprachelemente zur Folge; hierauf wird im einzelnen nicht besonders hingewiesen.

(1) Formaler Aufbau des Programms

- Die Sonderzeichen $ und : stehen nicht zur Verfügung.
- Maximal 9 Fortsetzungszeilen sind zulässig.
- Kommentarzeilen dürfen nur vor der ersten Zeile einer Anweisung stehen; vor Fortsetzungszeilen sind sie nicht zugelassen.
- Die Reihenfolge der Anweisungen ist strenger festgelegt:

<table>
<tr><td rowspan="6">Kommentare</td><td colspan="2">PROGRAM, FUNCTION, SUBROUTINE</td></tr>
<tr><td rowspan="5">FORMAT</td><td>IMPLICIT</td></tr>
<tr><td>andere Spezifikationen</td></tr>
<tr><td>DATA</td></tr>
<tr><td>Anweisungsfunktionen</td></tr>
<tr><td>Ausführbare Anweisungen</td></tr>
<tr><td colspan="2">END</td></tr>
</table>

(2) Datentypen, Variable, Felder

- Es gibt keine doppeltgenauen Größen.
- Es gibt keine komplexen Größen.
- Es gibt keine benannten Konstanten. Entsprechend entfällt die PARAMETER-Anweisung.
- Textvariable und Textfelder dürfen nur mit konstanter Länge vereinbart werden; insbesondere ist die Längenangabe (*) un-

zulässig.

- Teilketten können nicht angesprochen werden.
- Felder dürfen höchstens drei Dimensionen besitzen.
- Dimensionsbeschreiber müssen die Form d2 haben, d.h. der kleinste Index in jeder Dimension ist immer 1.

(3) Ausdrücke

- Die logischen Operatoren .EQV. und .NEQV. fehlen.
- Der Konkatenationsoperator // entfällt.
- Feldelemente und Funktionsaufrufe dürfen nicht als Operanden von Index-Ausdrücken verwendet werden.

(4) Steuerung des Programmablaufs

- Als Wert i, der bei einer berechneten GOTO-Anweisung die Auswahl des Sprungziels steuert, muß eine INTEGER-Variable angegeben werden; Ausdrücke sind nicht zugelassen.
- Bei DO-Anweisungen müssen sowohl die Laufvariable als auch alle angegebenen Parameter den Typ INTEGER besitzen.

(5) Unterprogramme

- Jedes Unterprogramm besitzt genau einen Eingangspunkt, der durch den Namen des Unterprogramms bezeichnet wird; die ENTRY-Anweisung entfällt.
- Außergewöhnliche Rücksprünge sind nicht zugelassen. Damit entfallen Marken als Parameter.
- BLOCK DATA-Unterprogramme entfallen.
- Der Typ CHARACTER ist für alle Funktionen unzulässig, also sowohl für externe Funktionen als auch für Anweisungsfunktionen.
- Für die inneren Standardfunktionen existieren keine Gattungsnamen; ihre Anzahl ist stark eingeschränkt. Zur Verfügung stehen nur

ABS	AMOD	FLOAT	LLE	SIGN
ACOS	ANINT	IABS	LLT	SIN
AINT	ASIN	ICHAR	MAXO	SINH
ALOG	ATAN	IDIM	MAX1	SQRT
ALOG10	ATAN2	IFIX	MINO	TAN
AMAXO	COS	INT	MIN1	TANH
AMAX1	COSH	ISIGN	MOD	
AMINO	DIM	LGE	NINT	
AMIN1	EXP	LGT	REAL	

- Bei den Vergleichsfunktionen für Texte müssen beide Parameter die gleiche Länge besitzen.

(6) Arbeiten mit Dateien, Formatierung

- Die PRINT-Anweisung und die Kurzform der READ-Anweisung entfallen.
- Ausgabelisten dürfen, wie Eingabelisten, keine Ausdrücke enthalten.
- Bei impliziten DO-Elementen müssen sowohl die Laufvariable als auch alle angegebenen Parameter den Typ INTEGER besitzen.
- Gerätenummern dürfen nur in der Form u ohne das Schlüsselwort UNIT= angegeben werden; sie müssen immer an erster Stelle in der Liste der Steuerparameter stehen. u muß ein Stern oder eine INTEGER-Konstante sein.
- Formate dürfen nur in der Form f ohne das Schlüsselwort FMT= angegeben werden; sie müssen bei formatierter Ein-/Ausgabe an zweiter Stelle in der Liste der Steuerparameter stehen. f muß die Anweisungsnummer einer FORMAT-Anweisung oder eine Textkonstante (keine Textvariable) sein; listengesteuerte Umwandlung ist nicht zugelassen.
- Die Steuerparameter ERR=s und IOSTAT=ios entfallen.
- Im Parameter REC=rn muß rn eine Konstante oder Variable sein. Ausdrücke sind unzulässig.
- Dateinamen können nicht angegeben werden; der Steuerparameter FILE=fn entfällt. Zuordnungen zwischen benannten Dateien und

Dateinummern können aus dem Programm heraus nicht erfolgen.

- Direkte Dateien müssen unformatiert sein.
- Textfelder dürfen als Ganzes nicht als interne Dateien angesprochen werden. Interne Dateien bestehen damit immer aus genau einem Satz.
- Bei der OPEN-Anweisung stehen nur die drei Steuerparameter u, ACCESS='DIRECT' und RECL=rl zur Verfügung. Diese drei Parameter müssen immer angegeben werden; sequentielle Dateien können nicht explizit eröffnet werden.
  Hierbei muß rl eine INTEGER-Konstante oder -Variable (kein Ausdruck) sein; die Dateinummer u darf bislang keiner anderen Datei zugeordnet sein.
- Die CLOSE-Anweisung entfällt. Dateien können also nur implizit durch die Beendigung des Programms abgeschlossen werden.
- Die INQUIRE-Anweisung entfällt.
- Die Anweisungen BACKSPACE, ENDFILE und REWIND stehen nur in der Kurzform zur Verfügung.
- Nicht zur Verfügung stehen die Formatbeschreiber Iw.m, Dw.d, Gw.d, Gw.dEe, T, TLc, TRc, :, S, SP und SS.

(7) Speicherorganisation

- Bei SAVE-Anweisungen dürfen nur Namen von COMMON-Blöcken angegeben werden, muß mindestens ein Name angegeben werden.
- Bei DATA-Anweisungen erfolgt keine Typumwandlung; Texte dürfen kürzer, nicht jedoch länger als die Textvariable bzw. als das Element eines Textfeldes sein, in das sie übertragen werden.
- Bei DATA-Anweisungen sind implizite DO-Elemente unzulässig.
- DATA-Anweisungen sind für Größen aus COMMON-Blöcken unzulässig.
- Durch EQUIVALENCE dürfen CHARACTER-Größen nur gleichgesetzt werden, wenn sie die gleiche Länge besitzen.

LITERATURVERZEICHNIS

[1] American National Standards Institute, Inc.: Programming Language FORTRAN. ANSI X3.9-1978 (Revision of ANSI X3.9-1966). New York, N.Y. 1978

[2] Brainerd, W. (ed.): FORTRAN 77. Communications of the ACM 21, 800 - 820 (1978)

[3] Brauch, W.: Programmierung mit FORTRAN. Stuttgart 1979

[4] Control Data Corporation: FORTRAN, Version 5, Reference Manual 60481300. Sunnyvale, Cal. 1980

[5] Deutsches Institut für Normung (DIN): Programmiersprache FORTRAN. DIN 66 027. Berlin 1980

[6] Digital Equipment Corporation: VAX-11 FORTRAN Language Reference Manual. Maynard, Mass. 1980

[7] Digital Equipment Corporation: VAX-11 FORTRAN User's Guide. Maynard, Mass. 1980

[8] IBM: VS FORTRAN Application Programming: Language Reference GC26-3986-0. White Plains, N.Y. 1981

[9] Sperry Rand Corporation: Sperry Univac Series 1100 FORTRAN (ASCII) Level 9R1 Programmer Reference UP-8244.1. 1979

[10] Wehnes, H.: FORTRAN 77. München Wien 1981

Stichwortverzeichnis

ACCESS 115,124
Anfangswerte 160
Anweisung 11
 - , ausführbare 12
 - , formale Beschreibung 16
 - , Format- 127
 - , nicht ausführbare 12
 - , Typ- 87
 - zur Dateiverwaltung 107
 - zur Datenübertragung 106, 116
 - zur Positionierung 107
Anweisungsfunktion 59,63
Anweisungsnummer 11f
ASCII-Code 11,93,166
ASSIGN 36
assignment statement, arithmetic 27
 - , logical 49
Ausdruck, arithmetischer 24
 - , konstanter arithmetischer 27
Ausgabedatei 103
 - , Standard- 103,108,118
Ausgabefeld 128
Ausgabeparameter 72

BACKSPACE 107,118
BLANK 113,115,126,131
BLOCK DATA 162
Block IF 50
Buchstabe 11

CALL 71
character 11
CLOSE 107,113f,121
Code 92
 - , ASCII 11,93,166
 - , EBCDIC 11,93,166
 - , zulässiger 93
COMMON 154
COMMON-Block 154
 - , benannter 154
 - , Länge 156
 - , unbenannter 154
COMPLEX-Konstante 20
CONTINUE 46

DATA 96,160
Datei 102
 - mit direktem Zugriff 105
Datei mit sequentiellem Zugriff 104
 - , Ausgabe- 103
 - , direkte 105,115,125,130
 - , Eingabe- 103
 - , externe 102
 - , formatierte 103,125
 - , interne 102,108,130,151
 - , leere 102
 - , Positionierung einer 104
 - , sequentielle 104,115,118, 125
 - , Standard-Ausgabe- 103
 - , Standard-Ein-/Ausgabe- 108
 - , Standard-Eingabe- 103
 - , Standard-Stanz- 108
 - , temporäre 114
 - , unformatierte 103,125
Dateiende 106,110,119
Dateiname 102
 - , Abfrage nach 123
Dateinummer 103,108,112,121
 - , Abfrage nach 123
Daten, arithmetische 19
Datensatz 101
DIMENSION 38
Dimension eines Feldes 38
Dimension, Größe einer 38
DIRECT 124
direct file 105
DO 41
 - , implizites 111
DO-Schleife 42
doppeltgenau 15
DOUBLE PRECISION-Konstante 20
Drucken 140
Durchlaufzähler 42

EBCDIC-Code 11,93,166
edit descriptor 127
Ein-/Ausgabe, Fehler bei der 109
 - , formatgesteuerte 102,105
 - , formatierte 101
 - , listengesteuerte 29,102, 105
 - , unformatierte 101
Ein-/Ausgabeanweisungen, all-

gemeine Form 107
- , Kurzform 107
Ein-/Ausgabeliste 29,107,110, 128
einfachgenau 15
Eingabedatei 103
- , Standard- 103,108,117
Eingabefeld 128
- , leeres 149
Eingabeliste 109f
Eingabeparameter 72
Eingangspunkt 60
ELSE 55
ELSE IF 55
ELSE IF-Block 55
ELSE-Block 55
END 12,31,110,116
END IF 50
ENDFILE 107,118
ENTRY 83
EOF-Satz 106,110,119
EQUIVALENCE 156,158
ERR 109,112,116,119,121,123
EXIST 124
EXTERNAL 79

Feld 37,111
- , aktuelles 75
- , formales 75
- , Größe 38
Felddefinition 38
Feldelement 38,110
- , Name eines 39
file 102
FILE 112,123
file, INQUIRE by 123
FMT 116
FORM 115,125
Format 127
- , leeres 143
Formatanweisung 127
Formatbeschreiber 128
- 'h...h' 139
- / 144
- : 143
- A 137
- Aw 137
- BN 131
- BZ 131
- Dw.d 131,134
- Ew.d 131,134
- Ew.dEe 131,134
- Fw.d 131,134
- Gw.d 131,135
- Gw.dEe 131,135
- Iw 131,134
- Iw.m 131,134
- kP 146
- Lw 136
- nH 139
- nX 130
- S 133
- SP 133
- SS 133
- Tc 130
- TLc 130
- TRc 130
- , Daten- 128
- , Steuer- 128
formatgesteuerte Ein-/Ausgabe 102
formatiert 101
formatierte Datei 103
FORMATTED 125
FORTRAN-Anweisungen 17
FORTRAN-Zeichensatz 11
Fortsetzungszeile 12
FUNCTION 87
function, intrinsic 61
Funktion 59
- , externe 66
Funktionsname, spezieller 62
Funktionsunterprogramm 60

Gattungsname 62
Gleitkommagrößen 15
global 28,67
GOTO, berechnetes 35
- , unbedingtes 35
- , zugeordnetes 36

Hauptprogramm 11
Hilfsprozeduren 60

IF, arithmetisches 28
- , logisches 49
IF-Block 50,54
IF-Level 54
IMPLICIT 22,87
Implikation 51
Index 38
Indexausdruck 39
Indexbereich 38
Indexgrenze 38
Innere Standardfunktion 61
INQUIRE 107,122

INTEGER-Konstante 19
INTRINSIC 81
intrinsic function 59
IOSTAT 109f,112,116,119,121, 123

Klammerpaar 24
Kommentarzeile 11f
Konkatenation 91,111
Konstante, arithmetische 19
 - , benannte 22
 - , symbolischer Name einer 21

Länge (*) 87,94,111
 - einer Textgröße 86
 - eines Satzes 101,125
Laufvariable 41
leere Datei 102
leeres Eingabefeld 149
Liste, Ein-/Ausgabe- 107
listengesteuerte Ein-/Ausgabe 102
listengesteuerte Umwandlung 147
lokal 28,63ff,68

Marke 12

NAME 124
Name einer Datei 102
 - , globaler 67
 - , lokaler 68
 - , symbolischer 20
NAMED 124
Namensregel 21
Nebeneffekt 72
NEXTREC 125
null value 149
NUMBER 124

OPEN 107,112
OPENED 124
Operator, arithmetischer 24
 - , Konkatenations- 91
 - , logischer 48

PARAMETER 21
Parameter, aktueller 61,63
 - , formaler 61,67
 - , - , einer Anweisungsfunktion 63
PAUSE 31
Positionierung 104,129
 - sequentieller Dateien 118
PRINT 29,106,118
PROGRAM 28
Programm, ausführbares 11
Programmeinheit 11
Prozedur 59
 - , externe 60
 - , formale 78
PUNCH 107,118

READ 29,106,116,118
REAL-Konstante 19
RECL 115,125
record 101
reellwertig 15
RETURN 82
REWIND 107,118

Satz 101
 - , EOF- 106,110,119
 - , formatierter 101,115
 - , Länge eines 101,115
 - , leerer 102
 - , unformatierter 101,115
Satznummer 105,116,125
SAVE 96,163
Schlüsselwort 13
 - CHARACTER 86
SEQUENTIAL 124
sequential file 104
Service Subroutinen 60
Skalenfaktor 146
Sonderzeichen 11
Sortierfolge 92
Spalte 11
Speicherabbildungsfunktion 40
Speichereinheit, numerische 14
 - , Zeichen 13
Spezifikationsanweisung 23
Standard-Ausgabedatei 103,118
Standard-Eingabedatei 103,117
Standard-Stanzdatei 108,118
Standardfunktion, innere 59, 61
Stanzdatei, Standard- 118
statement, executable 12
 - , nonexecutable 12
STATUS 114,121
Steueranweisung 28
Steuerparameter 107
Steuerzeichen, Vorschub- 140
STOP 31
string 86

SUBROUTINE 70
Subroutine 59
Subroutine-Unterprogramm 60, 70
substring 89

Teilkette 89,91,111,151
temporäre Datei 114
Text 86
Textausdruck 91f
Textausdrücke, Vergleich von 92
Texte als Parameter 98
Textfeld 86,91,98,102,151
 - , Teilkette eines 89
Textfunktion 87,91
Textgröße, Länge einer 86
Textkonstante 13,29,88,91
Textvariable 86,91,102,151
 - , mit Länge (*) 94
 - , Teilkette einer 89
 - , Wertzuweisung für 91
Textverarbeitung 86
Trennzeichen 147
Typanweisung 22,87
 - , logische 47
 - , für Texte 86

Übertragung, formatierte 101,116
 - , unformatierte 101,116

Umwandlung 26
 - , formatgesteuerte 102,127
 - , listengesteuerte 102,147
unformatiert 101
unformatierte Datei 103
UNFORMATTED 125
UNIT 108,112,116,119,121,123
unit identifier 103
unit, INQUIRE by 123
Unterprogramm 11
 - , externes 59
 - , internes 59
 - , rekursives 82

Variable 21,110
Vereinbarung 24
Vergleich von Textausdrücken 92
Vergleichsausdruck 14,47,92
Vergleichsoperator 47,93
Verzweigung 28
Vorschubsteuerung beim Drucken 140
Vorschubsteuerzeichen 140

Wahrheitswert 14
Wert einer Konstanten 19
Wertzuweisung für Texte 91
 - , arithmetische 27
 - , logische 49
Wiederholungsbereich 41
Wiederholungsfaktoren in Ausgabesätzen 150
 - in Eingabesätzen 150
 - in Formaten 142
Wort 14
WRITE 106,116

Zeichen 14
Zeichenposition 11,14
Zeile 11
Ziffer 11
Zugriff, direkter 105
 - , sequentieller 104

Menzel: BASIC in 100 Beispielen/Disketten-Version APPLESOFT
2. Aufl. 216 Seiten. Beilage: Diskette mit allen
BASIC-Programmen in APPLESOFT. DM 59,80

Ottmann/Widmayer: Programmierung mit PASCAL
256 Seiten. DM 16,80

Richter: Betriebssysteme
152 Seiten. DM 22,80

Richter: Logikkalküle
232 Seiten. DM 24,80

Schlageter/Stucky: Datenbanksysteme: Konzepte und Modelle
261 Seiten. DM 24,80

Schmidt: Digitalelektronisches Praktikum
2. Aufl. 238 Seiten. DM 15,80

Schmidt u.a.: Digitalschaltungen mit Mikroprozessoren
2. Aufl. II, 205 Seiten. DM 23,80

Schneider: Problemorientierte Programmiersprachen
226 Seiten. DM 23,80

Singer: Programmieren in der Praxis
176 Seiten. DM 18,80

Singer: Programmierung mit COBOL
3. Aufl. 304 Seiten. DM 16,80

Spaniol: Arithmetik in Rechenanlagen
Logik und Entwurf
208 Seiten. DM 24,80

Waldschmidt: Schaltungen der Datenverarbeitung
264 Seiten. DM 38,--

Wirth: Algorithmen und Datenstrukturen
2. Aufl. 376 Seiten. DM 28,80

Wirth: Compilerbau
2. Aufl. 94 Seiten. DM 16,80

Wirth: Systematisches Programmieren
3. Aufl. 160 Seiten. DM 22,80

Preisänderungen vorbehalten

Teubner Studienskripten
Datenverarbeitung

Verarbeitung arithmetischer Daten

Steuerung des Programmablaufs, Block IF-END IF-Konstruktionen

Unterprogramme

Textverarbeitung

Dateiverwaltung

Ein-/Ausgabe

Speicherorganisation

B.G. Teubner Stuttgart

ISBN 3-519-00089-x